

For Reference

NOT TO BE TAKEN FROM THIS ROOM

Ex libris
UNIVERSITATIS
ALBERTAE NSIS



For Reference

NOT TO BE TAKEN FROM THIS ROOM

THE UNIVERSITY OF ALBERTA
A JOB SHOP SCHEDULING PROBLEM

by



ROBERT DALE FREEMAN

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
OF MASTER OF BUSINESS ADMINISTRATION

FACULTY OF BUSINESS ADMINISTRATION AND COMMERCE

EDMONTON, ALBERTA

OCTOBER 21, 1968

THESIS
1968 (F)
72

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "A Job Shop Scheduling Problem," submitted by Robert D. Freeman in partial fulfilment of the requirements for the degree of Master of Business Administration.

ABSTRACT

The problem of scheduling a fairly large number of jobs through a production facility containing a small number of identical processors is explored. The application of an analytic procedure is studied, but it is shown to produce computational problems which are beyond the scope of the existing computer and are, therefore, not practical.

Heuristic scheduling rules are then developed and their relative merits tested using several measures of effectiveness. One rule is shown to be best according to each measure of effectiveness. The result is, knowing the quantity he wants to optimize, one can choose a rule and obtain a schedule which will give a near optimum according to that measure.

Algorithms were developed for testing and implementing the rules. The time shared APL system at the University of Alberta, which permits remote access to an IBM 360/67 computer, was used in the simulations.

ACKNOWLEDGEMENTS

I wish to express my appreciation to Dr. D. H. Bent who proposed this problem and for his valuable guidance in developing the solutions to it.

TABLE OF CONTENTS

		Page
CHAPTER 1	THE SCHEDULING PROBLEM	1
Section 1.1	Definition of the Scheduling Problem	1
Section 1.2	Example of the Scheduling Process	2
Section 1.3	Economic Significance	7
Section 1.4	Related Problems	9
Section 1.5	Summary of Results in This Thesis	14
CHAPTER 2	ANALYSIS OF THE SCHEDULING PROBLEM	19
Section 2.1	Formulation as an Integer Programming Problem	19
Section 2.2	Balas Algorithm	24
Section 2.3	Heuristic Methods	28
Section 2.4	Examples of Scheduling Rules	30
CHAPTER 3	EVALUATION OF SCHEDULING RULES	32
Section 3.1	Data Used to Evaluate the Rules	32
Section 3.2	Criteria to Measure Effectiveness of Scheduling Rules	37
Section 3.3	Application of the Measures of Effectiveness	42
Section 3.4	Ranking of the Scheduling Rules	44
Section 3.5	Comparison of the Scheduling Rules	53
Section 3.6	Unusual Rosters	74
Section 3.7	Reliability of the General Rules	87

TABLE OF CONTENTS (Cont'd.)

	Page
CHAPTER 4 PROGRAMMING OF SCHEDULING MODELS	90
Section 4.1 Outline of the APL System	90
Section 4.2 Programs Used in the Simulation	99
Section 4.3 Data Handling	124
 CHAPTER 5 FURTHER RESEARCH	 129
 BIBLIOGRAPHY	
 APPENDICES	

CHAPTER I

THE SCHEDULING PROBLEM

1.1 Definition of the Scheduling Problem

The scheduling problem is to allocate a set of jobs to a shop or production facility in such a way as to minimize or maximize some measure of effectiveness. For our purposes, a shop will consist of an integral number of units of capacity which will be available for a finite time. Time will also be expressed as an integral number of units. Thus, a shop will be specified by a vector C where the N 'th element $C[N]^*$ is the capacity available in time period N .

A job is defined by two parameters, capacity and duration. The capacity requirements of the J 'th job, $X[J]$, is the number of units of shop capacity required to process the job and is expressed in the same units as the capacity of the shop. The duration of the J 'th job, $P[J]$, is the number of units of time required to process the job and is expressed in the same units of time used to define the shop. Both capacity and duration are integers. We define the size of the job to be the product of its capacity and duration or $X[J] \cdot P[J]$.

The set of jobs to be processed, or job roster, is defined by the two vectors, capacity X and duration P . The J 'th job is defined by the J 'th element of each of these vectors. No jobs are added to the roster while the shop is processing the jobs it originally contained.

A schedule is defined by a vector NUM where $NUM[J]$ is the starting time for job J . It is the allocation of a roster of jobs through the shop. The schedule must be feasible; that is, in any time period,

* The APL notation used throughout is described in Section 4.1.3.

the total capacities of the jobs in process must not exceed the capacity available in the shop in that period. The scheduling process is not constrained by limitations on the number of time periods the shop is in operation.

If a job is scheduled to occupy a unit of capacity in a given time period, it will occupy the complete unit of capacity for the complete time period. The model may be visualized as a shop containing machines which can do only one job at a time and can only be set up for different jobs at certain intervals (e.g., each morning) making it impossible to change jobs in the time period (during the day).

A measure of effectiveness is a criteria for evaluating a schedule. It is a mathematical formulation of some characteristic of the schedule which is considered to be important and becomes the objective function in the analytic approach to the problem. For example, if it is important to process many jobs as soon as possible, mean start time would be a suitable measure of effectiveness.

The following example shows the problem and the scheduling process for a simple case.

1.2 Example of the Scheduling Process

Example Problem 1 below illustrates the allocation of a roster of three jobs to a given production facility or shop. The shop and rosters to be scheduled are first defined.

The shop is to have five units of capacity and run for 25 time periods. It is represented by the vector $C=5,5,\dots,5$ or 25_5 . The jobs are defined by two vectors:

$$X = 2,4,3$$

$$P = 1,3,6$$

This defines three jobs; the first occupies two units of capacity for one unit of time represented by

$$X[J]= 2, \quad P[J]= 1 \quad J = 1$$

The second and third jobs occupy four capacity units for three time units and three capacity units for six time units, respectively, and are represented as follows:

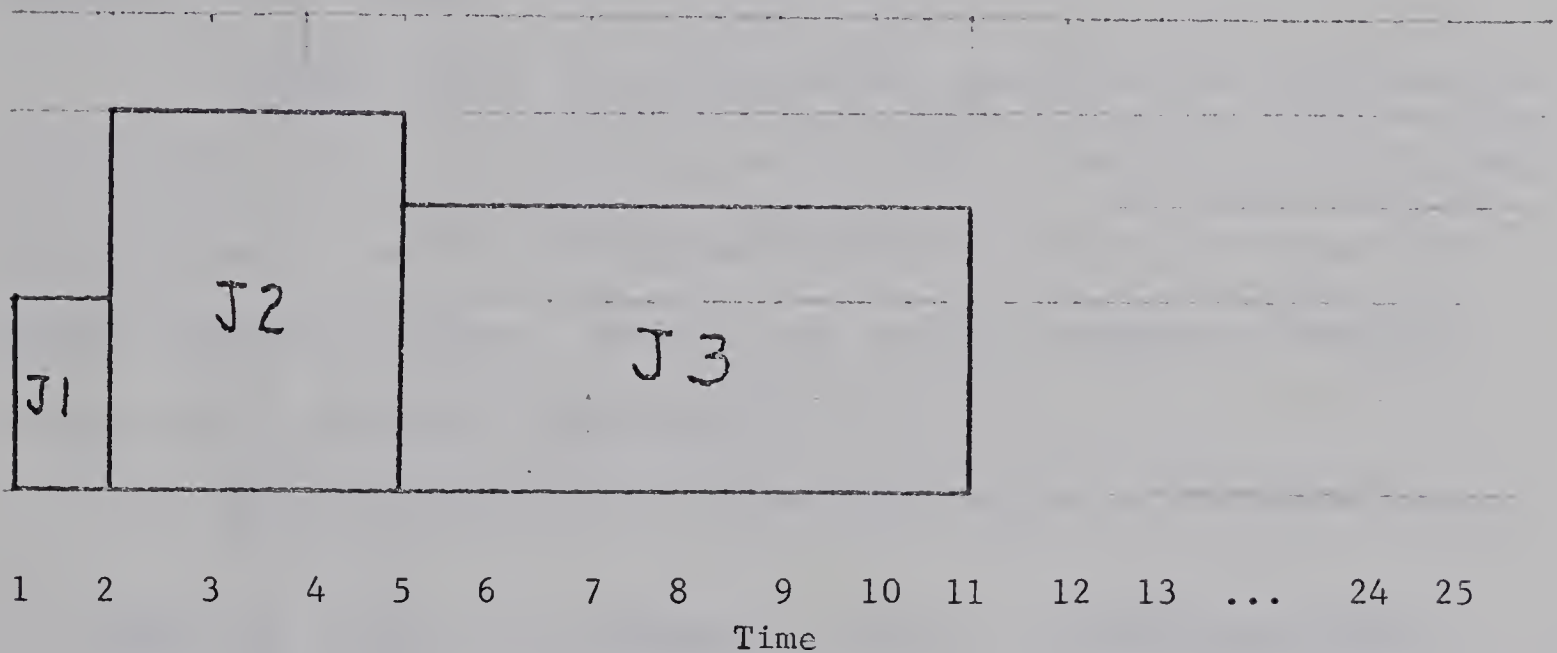
$$X[J]= 4, \quad P[J]= 3 \quad J = 2$$

$$X[J]= 3, \quad P[J]= 6 \quad J = 3.$$

Figure 1 shows the shop and the allocation of the jobs to it on a "first-come-first-served" basis.

Figure 1

First-Come-First-Served Assignment of Example Problem 1



The schedule is defined by the vector NUM which has the following value:

$$NUM = 1, 2, 5$$

This means job 1 starts in period 1, job 2 starts in period 2, and job 3 starts in period 5. The finish times are related to the start times and are represented by the vector F:

$$F = 1, 4, 10$$

The number of units of time the shop is occupied is defined as

Max $F[J]$ or 10 in this case. The unused shop capacity after the $1 \leq J \leq J_{\max}$

schedule is determined is $C[N]$ $1 \leq N \leq \text{Max. } F[J]$

The number of jobs in process is represented by a vector JOB where JOB N is the number of jobs in process in time period N. For this case

$$\text{JOB}[N] = \begin{cases} 1, & 1 \leq N \leq 10 \\ 0, & N > 10 \end{cases}$$

The sum of start times is $J = \sum_{J=1}^{J_{\max}} \text{NUM } J$ or $+/\text{NUM}$, that is,

$1 + 2 + 5 = 8$ for this schedule. The average start time is this quantity divided by J_{\max} or $8/3$.

Another measure of effectiveness is based on the idea that the profit from a job is proportional to its size. A delayed profit is then proportional to job size multiplied by delay or units of elapsed time before processing begins. This is also called a weighted disutility factor and is described subsequently in 3.2.1.f.

For this schedule, the disutility factor is calculated below.

Job	Delay	Size	Disutility Factor
1	0	22	0
2	1	12	12
3	4	18	72

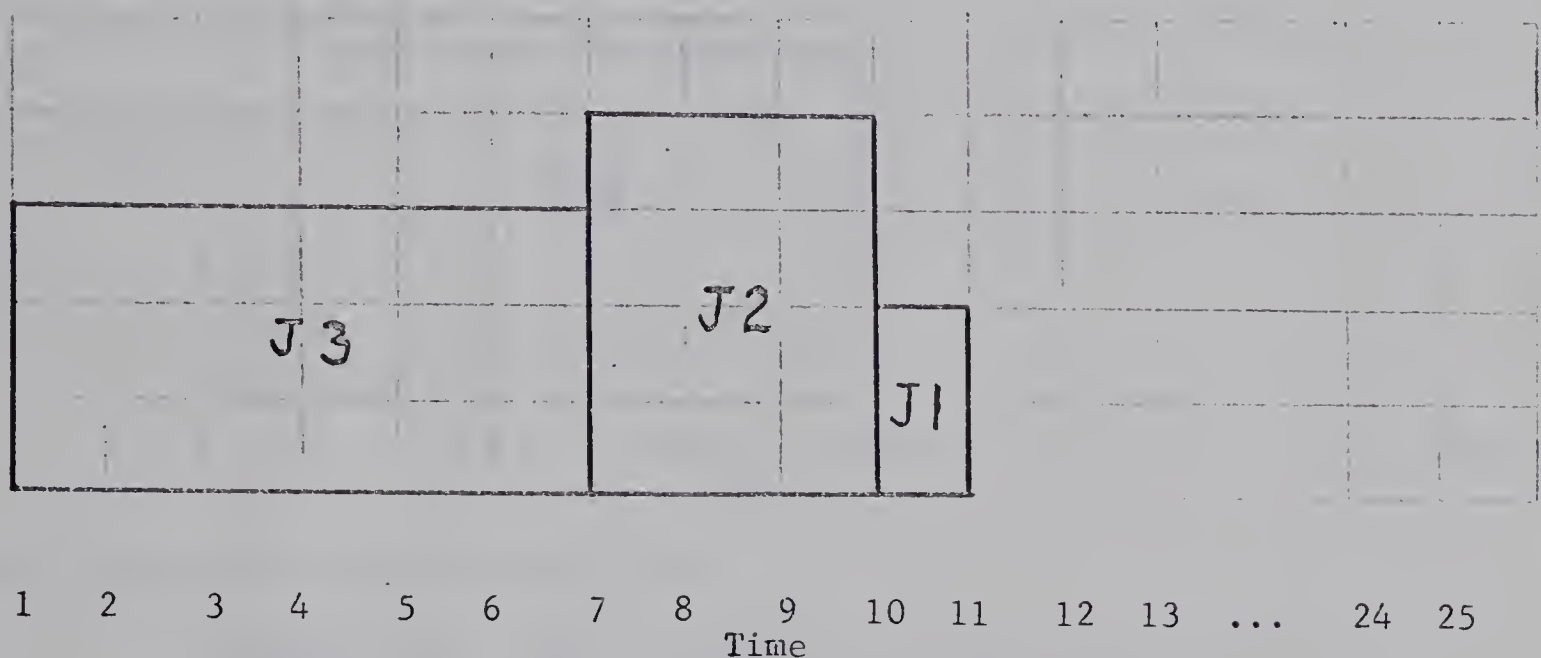
Total delayed profit or disutility factor is $0 + 12 + 72 = 84$.

More complex rules may be used to schedule the jobs resulting in different schedules. The calculated value of each measure of

effectiveness will then be different than that calculated above; in fact, the numerical value of each measure of effectiveness is a function of the schedule and therefore depends on the scheduling rule. A heuristic rule may be used: take the largest remaining job first. Figure 2 shows the schedule resulting from applying this heuristic rule to the roster and shop in the example problem.

Figure 2

Largest Remaining Job Scheduled First - Example Problem 1



This schedule is defined by the start time vector

$$\text{NUM} = 10, 7, 1$$

and the related finish time vector is

$$\text{F} = 10, 9, 6$$

The three measures of effectiveness are mean start time which is $(+/\text{NUM}) \div 3 = (10 + 7 + 1) \div 3 = 6$, shop occupied time = Max F which is F = 10 and a delayed profit of $(9 \times 2) + (6 \times 12) + (0 \times 18) = 90$.

The next heuristic rule applied to this example problem is similar to the above rule except that the largest capacity job will be scheduled first, and an attempt will be made to fill empty spaces with

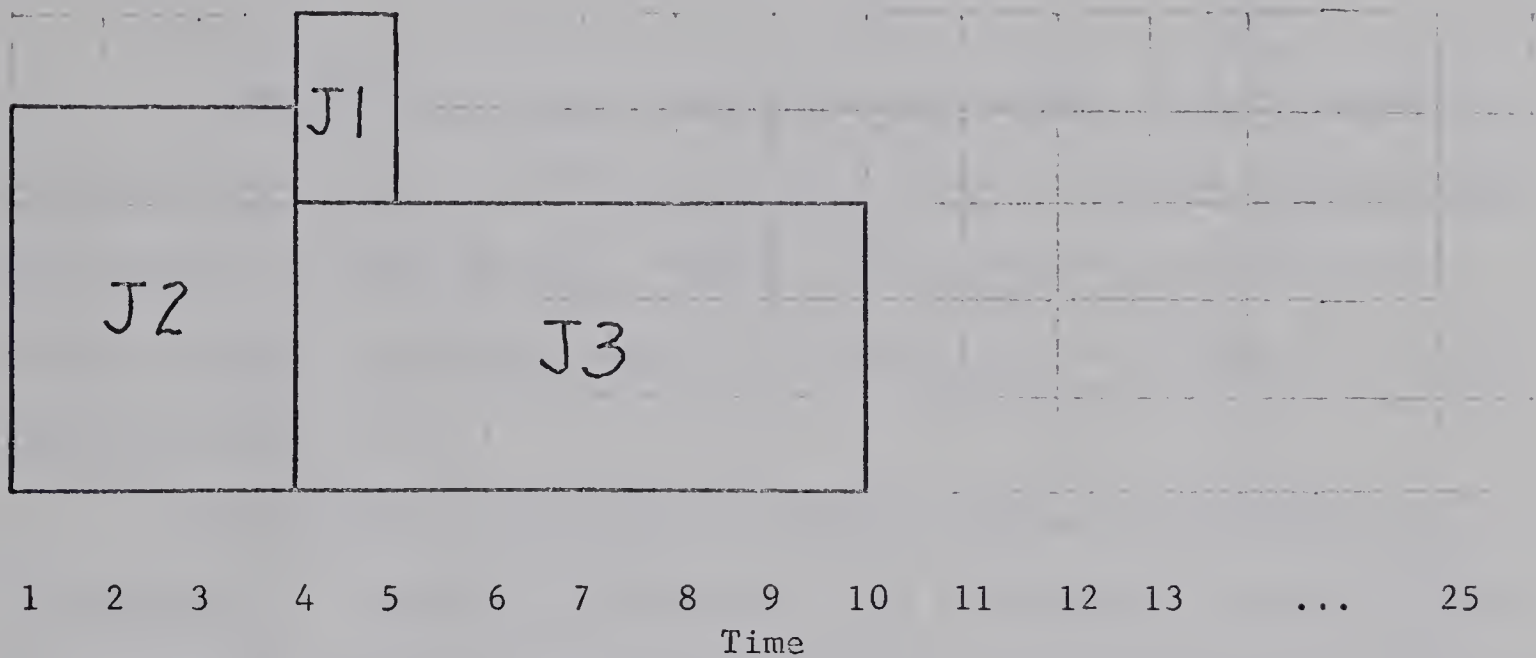
smaller capacity jobs. This resulting schedule is represented by Figure 3 and the vectors:

$$\text{NUM} = 4, 1, 4$$

$$\text{F} = 4, 9, 3$$

Figure 3

Largest Capacity First and Fill Empty Spaces - Example 1



The measures of effectiveness are:

$$\text{Average start time} = (+/\text{NUM}) \div 3 = (4 + 1 + 4) \div 3 = 3$$

$$\text{Shop occupied} = \max \text{F} = 9$$

$$\text{Delayed profit} = (2 \times 3) + (0 \times 12) + (3 \times 18) = 60$$

The values of each measure of effectiveness for each scheduling rule are summarized in Table 1 below.

TABLE 1

Summary of Results - Example Problem 1

Schedule	Measure of Effectiveness		
	Shop Occupied	Mean Start Time	Delayed Profit
1	10	8/3	84
2	10	6	90
3	9	3	60

Schedule 2 is a poor arrangement according to these measures of effectiveness as it yields the poorest value of all the schedules for each measure of effectiveness. However, it is entirely possible that another measure of effectiveness is of importance; and Schedule 2 ranks high according to it.

The example illustrates a conflict between the measures of effectiveness. Considering Schedules 1 and 3, Schedule 1 gives a better average start time while Schedule 3 gives a better delayed profit loss and reduces the time the roster occupies the shop. Either Schedule 1 or 3 will therefore be considered best depending on whether average start time or delayed profit loss is considered by the scheduler to be more important.

1.3 Economic Significance

Our objective is to develop a practical scheduling technique which may be used in a job shop. To be practical, the technique must use equipment and time only in such quantities as are justified by the savings resulting from using it. Also, the technique must be fast enough so that the information gathered from its use is available in time to be usable.

It must also be possible for the scheduler to choose the quantity that is important to him as a measure of effectiveness. This requires that important measures of effectiveness must be expressed in formula form if an analytic technique is chosen; or if heuristic methods are used, a particular heuristic rule in a set of rules must be shown to be associated with certain behavior measured by each measure of effectiveness.

A job shop typically consists of more than one department, and each job travels through the shop visiting each department in a given sequence. The sequence of movements between departments and time spent in each department is usually different for each job.

The current study focuses on one department. There is a queue or roster of jobs waiting for service at that department, and no new jobs are added to the queue until the entire queue or roster has been processed by the department. The department can handle more than one job at a time as long as the combined capacity requirements of all jobs running do not exceed the capacity of the department.

While this is a simple problem, it is the key to the entire job shop scheduling problem; that is, the individual department. Behavior within the department must be understood before we can analyze flows between the departments. This study gives some results for a static problem where no new jobs enter the roster during processing; however, the model developed is sufficiently general that a dynamic problem involving arrivals to the queue could be studied. The conclusions of this study should provide some insight into what would happen in a dynamic case.

This study, therefore, is a start on the solution of the

problem of allocating jobs with fixed capacity and duration requirements through a shop with fixed capacity in each department. The time period and unit of capacity which the job must occupy entirely or not occupy at all is large relative to the size of the job meaning a good "fit" between the jobs in the shop is required to avoid losing large amounts of machine time.

This model is reasonable as we could visualize a department containing a few high priced identical machines which require considerable set-up time for each job. Each job requires one or more machines for processing. A problem where there are many facilities and small set-up time has more of a flow or assembly line nature and is not really a job shop in the sense discussed here.

1.4 Related Problems

A model often analyzed in the literature consists of a job shop containing a number of different machines or departments and jobs which follow a predetermined sequence of steps through the shop. [3,4,5] The jobs are defined by one dimension, duration, and occupy the entire department when they are in it.

This is in contrast to the model discussed above where the entity studied is one department. In many models, the department is indivisible; in this case, it contains nine identical units of capacity. Each job occupies one or more units of department capacity for a given number of units of time.

Analysis of the scheduling problem using both analytic and heuristic methods has been presented in the literature. Due to analytic methods becoming impractical because of the volume of computational

requirements, major attempts have been made to develop heuristic methods which give good approximations to optima for certain measures of effectiveness.

An analytic approach to a job shop scheduling problem is presented by Brooks and White.[3] The problem consists of several machines; each job is routed around the machines in a given sequence. In the solution, a matrix A is developed which is equivalent to the A matrix in the standard zero - one integer programming problem form:

$$\min cx$$

$$\text{subject to } Ax \leq b ; x = 0 \text{ or } 1$$

To develop this matrix, two other matrices are developed and combined:

1. A sequencing array consisting of the routing order of each job around the machines.
2. A facility array consisting of the time each job spends on each machine.

The problem array, or A matrix, indicates the minimum completion time for each job on each machine and is derived from the above two arrays.

The integer programming formulation could be completed by expressing the objective function or measure of effectiveness as the c vector and introducing capacity constraints in the b vector.

However, the author states that this formulation becomes too massive if a problem of practical size is considered and could not be implemented on existing computers. Ten small problems were done using this method, giving an optimum value for three different objective functions, and the results are then compared with the results from application of three heuristic scheduling methods.

The three heuristic rules are:

1. Random selection
2. Shortest operation first
3. Longest remaining time to completion first

The three measures of effectiveness are:

- A. Lateness
- B. Machine idle time
- C. Total time to complete all orders

Table 2 shows the comparison of the merits of using each of the four scheduling methods as indicated by each measure of effectiveness. A low number always indicates a better result than a high one, and the numbers represent dimensioned quantities rather than dimensionless ranks. A ranking method is described in Section 3.4.

TABLE 2

Comparison of Brooks and White Results

<u>Rule</u>	<u>Measure of Effectiveness</u>		
	A	B	C
Integer Programming	4836	3333	1180
Heuristic 1	8377	5586	1473
Heuristic 2	7108	5521	1422
Heuristic 3	9531	4911	1323

This result shows that for at least some measures of effectiveness, heuristic rules can be developed which produce results not far from optimum.

Using a rather extensive model, Conway has studied the effect of a large number of heuristic rules measured by several measures of

effectiveness.[4] The situation is similar to that studied by Brooks and White consisting of nine independent machines. The jobs need not be processed on all machines and may return to a given machine after a subsequent operation on another machine. The number of operations per job is a random variable with mean 9, and there is equal likelihood of the job going to any other operation after any one is completed. The time to perform each operation on each job is exponentially distributed with mean 1.

The scheduling rules were programmed and large rosters were processed by assigning priority on the basis of the following criteria or linear combinations of these criteria.

1. Random selection.
2. First-come-first-served.
3. Shortest or longest next process time.
4. Greatest or smallest number of remaining operations.
5. Largest or smallest total work required.
6. Queue which job enters for next operation:
 - a. Shortest queue.
 - b. Queue with least work required.

The measures of effectiveness used are indicators of work in process inventory and were:

1. Work remaining and not completed.
2. Work in process in the shop.
3. Work content in queues.

It was observed that there are important differences between rules that are simple and easily implementable, and different rules give different performances according to each measure of effectiveness. No one rule

is best according to all measures of effectiveness.

Fabrycky and Shamblyn develop and test a particular heuristic rule with a model similar to that described above.[5] The heuristic method is based on an urgency factor which is the probability that a job will be completed by its due date given the current conditions.

The total expected time for the i 'th order to complete its routing on n machines is

$$T_i = \sum_{j=1}^n t_j$$

where t_j is the time on the j machine and consists of

$$t_j = m_j + q_j + s_j + p_j$$

representing the move time, queue time, set-up time, and process time, respectively, associated with the j 'th machine.

T_i is a random variable whose value is affected by the multitude of factors affecting the values of the individual m , q , s , and p and the number of these quantities making up t_i for any particular job. By the Central Limit Theorem, we can say T_i has an approximately normal distribution with fixed mean and standard deviation.

Since T_i has a normal distribution at each point in time, a normalized Z factor can be calculated for each job.

$$Z_i = \frac{(D_i - C) - \sum_{j=k}^n \sigma_j}{\sqrt{\sum_{j=k}^n \mu_j^2}}$$

where D_i = due date of i 'th job

C = current date

$$\sum_{j=k}^n \sigma_j = \text{mean time to completion}$$

$$\sum_{j=k}^n \mu_j = \text{variance of the time to completion.}$$

Therefore, Z_i is a measure of the probability that the i 'th order will be completed by its due date.

The algorithm consists of computing Z_i for all jobs in each time period and scheduling those with the least probability of completion or greatest urgency factor. The urgency factor of the i 'th job is the Z_i calculated above. The measure of effectiveness is the percent of orders completed late, and the algorithm reduced this by 22 percent compared to a first-come-first-served routine.

1.5 Summary of Results in This Thesis

Fourteen scheduling rules were developed and their effect studied according to four measures of effectiveness. The scheduling rules were applied to sixty rosters of twenty-five jobs each. Half of the rosters were generated by a process which produces numbers uniformly distributed between two bounds, and the other half were generated by a Poisson process with fixed mean. Certain undesirable results were then eliminated from the Poisson rosters as described in Section 3.1.

1.5.1 The Scheduling Rules

The scheduling rules assign priority to job characteristics, that is according to largest or smallest capacity, duration, or size. In order to form a unique priority order, two assignments, called a primary sort and tie breaking or secondary sort, must be made. Some of the rules involve only one sort and are called primary scheduling rules. These assign priority according to one job characteristic only, and ties are broken in a random

manner. The rules which form a unique priority order are called secondary scheduling rules.

Except for the first scheduling rule in Table 3, a fitting procedure was implemented. If at any time the occupied capacity is less than shop capacity and occupied capacity plus capacity of the highest priority unassigned job exceeds shop capacity, a search is made through the remaining unassigned jobs in decending priority order for the first job that will not violate shop capacity. The first scheduling rule assigns on a first-come-first-served basis.

Table 3 shows the names of the scheduling rules and the priorities associated with each.

TABLE 3

Names of Scheduling Rules

Rule Name	Priorities	
	Primary Priority	Tie-Breaking Priority
RANDOM	first-come first-served	
RANDOMFIT	first job in roster	
SS	smallest size	
LS	largest size	
SP	smallest duration	
LP	largest duration	
SPSC	smallest duration	smallest capacity
SPLC	smallest duration	largest capacity
LPSC	largest duration	smallest capacity
LPLC	largest duration	largest capacity

Rule Name	<u>Priorities</u>	
	Primary Priority	Tie-Breaking Priority
SSSP	smallest size	smallest duration
SSLP	smallest size	largest duration
LSSP	largest size	smallest duration
LSLP	largest size	largest duration

These criteria do not necessarily determine the order in which the jobs appear in the final schedule. They determine the order in which an attempt is made to schedule the jobs subject to capacity constraints and the fitting procedure described above.

1.5.2 The Measures of Effectiveness

The measures of effectiveness used are defined as follows. The definitions are expanded in Section 3.2.1.

1.5.2.a Measure A or mean start time. This is the sum of the start times divided by the number of jobs. Where NUM is the vector of start times and NUM is the number of elements in this vector, mean start time is $(\sum \text{NUM}) / \text{NUM}$.

1.5.2.b Measure B or units of time the shop is occupied.

This is the maximum element of F, the vector of finish times, and is denoted by $\max F$.

1.5.2.c Measure C or average efficiency. This is the portion of available time used in the second and third quarters of the period of time the shop is occupied.

1.5.2.d Measure D or disutility factor. This is the size of each job multiplied by the time it is delayed before processing begins summed over the whole roster.

1.5.3 Results of the Study

The results of this study are summarized in Table 4 where the ranking of each of the four primary scheduling rules as measured by each measure of effectiveness are shown. The rankings are based on the test sample of sixty rosters. A rank of 1 indicates the best result and a rank of 4 indicates the poorest result indicated by the measure of effectiveness.

TABLE 4
Summary of Results

Scheduling Rule	<u>Measure of Effectiveness</u>			
	A	B	C	D
SS	2	4	4	4
LS	4	1	1	1
SP	1	3	3	3
LP	3	2	2	2

The conclusion drawn from this table is that if we wish to minimize mean start time, we give priority to jobs with the smallest duration requirement. To clear all jobs out of the shop in minimum time, maximize efficiency, or minimize the delayed profit factor, we give priority to the largest size job.

We have also found that certain tie-breaking sorts will improve the schedule as indicated by these measures of effectiveness. When priority is given to the smallest duration job which is necessary to minimize mean start time, ties should be broken by giving priority to the smallest capacity (or largest size) job to further improve the mean start time. When priority is given to the largest

size job, the other three measures are optimized further by giving priority to the smallest duration (or largest capacity) job in case of ties in the size priority assignment. In Section 3.5.2, it is shown that these relationships are statistically significant (0.05 level of significance or better).

The capacity of the shop was constant throughout the scheduling period. The schedule was constrained by the requirement that the sum of the capacities of the jobs in process must at no time exceed the capacity of the shop; however, the shop was allowed to run for a sufficient amount of time so that the schedule was not constrained by the amount of time the shop is running.

1.5.4 Summary

One rule in a set of simple rules based on roster characteristics (capacities, durations and sizes of jobs) has been shown to produce a best result according to each of the four measures of effectiveness. Conflicts regarding which scheduling rule is best exist as a different rule has been shown to be best depending on the measure of effectiveness which is held to be important.

The rule which has shown to be the best in the set has not been shown to be optimal or near optimal; however, a more optimal rule would be more complex and difficult to apply in practice. It is felt that the set of rules is reasonably comprehensive, and one rule in a comprehensive set of rules is likely to produce a near optimum according to any measure of effectiveness. An indication of the ability of the rules to produce optima is given by mid-range efficiency, as the optimal is known to be 1; and for many rosters, the efficiency produced by the best rule was 1.

ANALYSIS OF THE SCHEDULING PROBLEM

2.1 Formulation as an Integer Programming Problem

We will now present an analytic method which could be used to solve the scheduling problem. To formulate it as an integer programming problem, we must express it in the form:

$$\begin{array}{ll} \min & cy \\ \text{subject to} & Ay \leq b \\ & y = 0 \text{ or } 1 \end{array}$$

This is the standard zero-one integer programming form except the variable x is replaced with y to avoid confusion with x denoting the capacities of the jobs.

The zero-one variable y is defined:

$$y_{j,n} = \begin{cases} 1 & \text{if job } j \text{ starts at time } n \\ 0 & \text{if job } j \text{ does not start at time } n \end{cases}$$

The first part of the A matrix represents the constraint that a job starts once and only once:

$$\sum_{n=1}^N y_{j,n} = 1 \quad (j = 1, 2, \dots, J)$$

where there are J jobs and N periods of time available.

The remainder of the constrain matrix A specifies that the total capacity requirements of all jobs running in each time period must not exceed the shop capacity in that time period. The constraint is:

$$\sum_{j=1}^J y_{j,n} x_j \leq c_n \quad (n = 1, 2, \dots, N)$$

The x_j corresponds to $X [J]$ defined in Section 1.1 as the capacity requirement of the j 'th job, and c_n is the $C [N]$ representing shop capacity in period n .

Similarly, each of the first J rows of the matrix represents this relationship with j equal to the row index.

The last of the constraint matrix or the $(J + 1)$ to $(J + N)$ rows represent the capacity constraints.

Evaluating the first row, we find

$$x_1 y_{1,1} + x_2 y_{2,1} + \dots + x_J y_{J,1} + 0 + \dots + 0 \leq c_1$$

or in summation form

$$\sum_{j=1}^J x_j y_{j,n} \leq c_n \quad n = 1$$

In the second and subsequent periods of time, allowances must be made for jobs which have been started in an earlier time period and have extended into the current period. This is done by defining an \bar{x}_j as a column vector of elements x_j and order p_j . \bar{x}_j , therefore, extends downward in the A matrix (direction of increasing n or time) through a number of rows equal to p_j , the job duration or the time it must run once it is started.

The objective function cy of the integer programming formulation is used to introduce a measure of effectiveness into the problem. The vector c represents the objective function as shown in two examples below.

If we want to minimize mean start time, we minimize the sum

$$\sum_{j=1}^J \sum_{n=1}^N n y_{j,n}$$

or its equivalent cy in matrix notation where

$$c = [1, 1, \dots, 1 \quad 2, 2, \dots, 2 \quad n, n, \dots, n]$$

$$y^T = [y_{1,1} \ y_{2,1} \dots y_{J,1} \ y_{1,2} \ y_{2,2} \dots y_{J,2} \ y_{1,N} \ y_{2,N} \dots y_{J,N}]$$

If we want to minimize the measure of delayed profit, we introduce the size of the jobs in the formulation.

$$\sum_{j=1}^J \sum_{n=1}^N y_{j,n} \cdot s_j$$

where $s_j = x_j \cdot p_j$ or the size of the j 'th job. In the formulation c , c becomes

$$c = [s_1, s_2, \dots, s_J, 2s_1, 2s_2, \dots, 2s_J, \dots, Ns_1, Ns_2, \dots, Ns_J]$$

An example of the formulation of a simple problem will now be given before a method of solving the problem is described.

2.1.3 Example

Example Problem 1, Section 1.2, is formulated as an integer programming problem using this method. First we observe that the roster can be completed in ten units of time even with a very poor schedule. Therefore, $N \geq 10$ and we can choose $N=10$ for simplicity.

For the constraint or A matrix, we construct the first J rows noting $J=3$ as there are three jobs in the roster. Since $N=10$, ten unit matrices of order three arranged horizontally are required.

To construct the last $N=10$ rows, observe that for:

$j = 1$	$x_j = 2$	$p_j = 1$
$j = 2$	$x_j = 4$	$p_j = 3$
$j = 3$	$x_j = 3$	$p_j = 6$

Therefore, the extended vectors representing job duration as well as capacity become:

$$\bar{x}_1^T = [2]$$

$$\bar{x}_2^T = [4, 4, 4]$$

$$\bar{x}_3^T = [3, 3, 3, 3, 3, 3]$$

The constraint matrix is as follows where a blank (-) indicates that the cell would be a zero regardless of the particular problem.

1--	1--	1--	1--	1--	1--	1--	1--	1--	1--
-1-	-1-	-1-	-1-	-1-	-1-	-1-	-1-	-1-	-1-
--1	--1	--1	--1	--1	--1	--1	--1	--1	--1
243	---	---	---	---	---	---	---	---	---
043	243	---	---	---	---	---	---	---	---
043	043	243	---	---	---	---	---	---	---
043	043	043	243	---	---	---	---	---	---
003	003	043	043	243	---	---	---	---	---
003	003	003	043	047	243	---	---	---	---
000	003	003	003	043	043	243	---	---	---
000	000	003	003	003	043	043	243	---	---
000	000	000	003	003	003	043	043	243	---
000	000	000	000	003	003	003	043	043	243
000	000	000	000	000	003	003	003	043	043

The b vector in the constraint $Ay \leq b$ is

$$b^T = [1 \ 1 \ 1 \ 5 \ 5 \ 5 \ 5 \ 5 \ 5 \ 5 \ 5 \ 5 \ 5]$$

meaning that the shop has five units of capacity in each of ten periods of time.

The specific y vector of zero one variables is

$$y^T = [y_{1,1} + y_{2,1} + y_{3,1} + \dots + y_{1,10} + y_{2,10} + y_{3,10}]$$

The constraint $Ay \leq b$ therefore becomes:

1) for the one start only constraint

$$\sum_{n=1}^{10} y_{j,n} = 1 \quad (j = 1, 2, 3)$$

2) for the capacity constraints

$$2y_{1,1} + 4y_{2,1} + 3y_{3,1} \leq 5 \quad (n = 1)$$

$$3y_{3,5} + 3y_{3,6} + 3y_{3,7} + 4y_{2,8} + 3y_{3,8} + 4y_{3,9} +$$

$$3y_{3,9} + 2y_{1,10} + 4y_{2,10} + 3y_{3,10} \leq 5 \quad (n = 10)$$

The objective functions of mean start time or delayed profit can be introduced into the formulation using the c vector. For mean start time:

$$c = [1, 1; 1, 2, 2, 2, \dots, 10, 10, 10]$$

and for delayed profit where $S = 2, 12, 18$

$$c = [2, 12, 18, 4, 12, 36, \dots, 20, 120, 180]$$

2.2 Balas Algorithm

2.2.1 Description of the Algorithm

The Balas Algorithm provides a possible solution to this problem. [2] Any problem expressed in the form

$$\min cy$$

$$\text{subject to } Ay \leq b \quad \text{and } y = 0 \text{ or } 1$$

is a zero-one integer programming problem and is solvable using an additive algorithm due to Balas. Our problem has been expressed in this form.

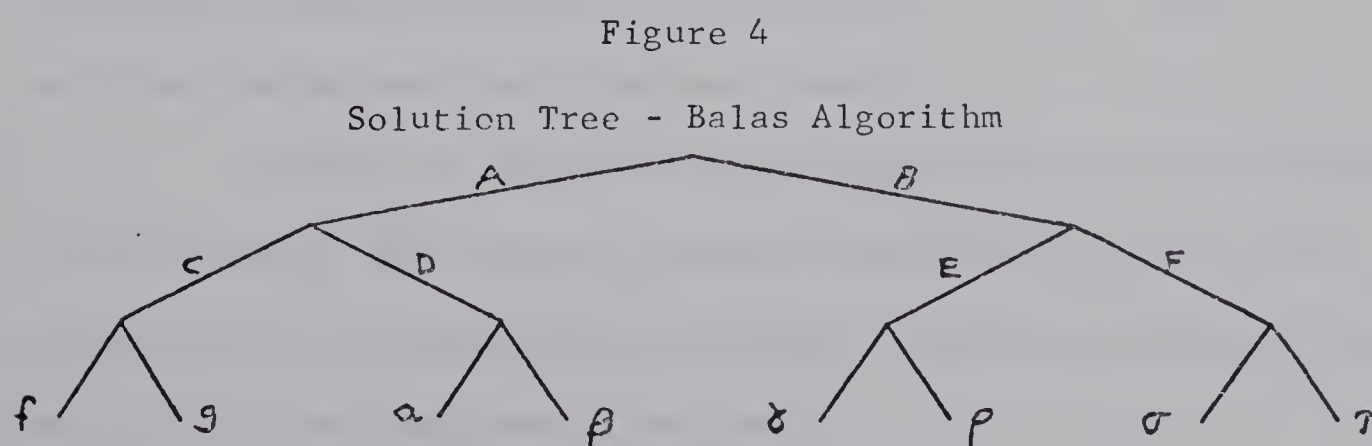
Since there are a finite number of ways a fixed number of jobs can be scheduled through a shop given the capacity restriction, there are a finite number of feasible solutions to

this problem. These are the solutions satisfying the constraint $Ay \leq b$. There may also be one or more optimal solutions representing schedules which meet the constraints and minimize the objective function.

A zero-one integer programming formulation is needed because y is constrained to be 0 or 1, not any positive value as in linear programming. In this problem, the shop capacity is divided into a small number of indivisible units whose capacity is important relative to the total shop capacity. For this reason, a variable representing an "either on or off" situation rather than a continuous flow is needed representing the fact that a job is either completely in or out of a capacity unit.

The Balas Algorithm starts by setting all n zero-one y variables to zero and by systematically assigning certain variables the value 1, a feasible optimal solution is obtained or evidence is obtained that no feasible solution exists. This is accomplished by trying only a small set of the possible 2^n combinations of $y = 0$ or 1.

Figure 4 shows the solution tree for the Balas additive algorithm where each horizontal plane of intersections represents y_1, y_2, \dots as we proceed down the diagram.



If the line connecting any variable to the previous variable or the start point approaches the plane representing the variable from the northeast, the variable is assigned the value 0. Approach from the northwest indicates assigning the variable the value 1. For example, branch A approaches the y_1 plane from the northeast and, therefore, indicates $y_1 = 0$. The end point 'a' represents the solution $y_1 = 0, y_2 = 1, y_3 = 0$. All eight ways of assigning $y_1, y_2, y_3 = 0$ or 1 are represented by the eight terminal points on the last plane of the diagram.

If each end point were tested for feasibility and the optimum solution or solutions picked out, we would eventually solve the problem. However, this would be time consuming on a large problem; and the Balas Algorithm identifies branches that if we proceeded down to the end of, we would not find any feasible solutions or a feasible solution better than the best one that has already been found. For example, if neither solution α or β is feasible, this is recognized at the point of intersection of branch A and D and branch D is "cut off."

If a feasible solution f has been found and solutions δ, ρ, σ , and τ are also feasible but not as good as f , this is recognized at the intersection of branch A and B so the entire right side of the diagram would be cut off and not tried. The only solutions which would be tried are f and g .

Balas has shown that there is no chance that a feasible solution better than the one already obtained will be cut off. The method, therefore, guarantees an optimal feasible solution to the zero-one integer programming problem.

2.2.2 Applicability of the Algorithm

One of the factors affecting the practicability of the use of the Balas Algorithm for this problem is the size of the arrays required. These are:

Vector c - objective function - length JN

Vector b - constraint vector - length $J + N$

Matrix A - constraint matrix - JN

$J + N$ rows

Therefore, the total number of elements required to formulate the problem in integer programming form is $J + N + JN + JN(J+N)$ in addition to JN zero-one variables.

For the problem to have any practical significance and to have different scheduling rules produce different schedules for comparison, considerably larger rosters than the one of size 3 in the example are needed. For any given measure of effectiveness, it is evident how to schedule a small roster; but we want a practical scheduling technique for a roster where the optimum is not obvious.

Rosters of twenty-five jobs with average capacities one-third to one-half of shop capacity were chosen as representative of the situation we wish to simulate. A roster of this size will occupy the shop for about sixty units of time when the jobs are assigned on a first-come-first-served basis.

Therefore, in the notation above, $J = 25$ and $N = 60$. The size of the input matrices for a worthwhile problem becomes:

c - objective function - $JN = 25 \times 60 = 1,500$ cells

b - constraint vector - $J + N = 25 + 60 = 85$ cells

A - constraint matrix - $(JN)(J+N) = 1,500 \times 85 = 127,500$
cells

The elements in these cells form an orderly pattern, and it is possible to program the computer to generate these arrays given the roster and the shop. However, a severe storage problem in the computer develops as explained in Section 4.1. In addition, the algorithm is very slow when the input A matrix exceeds size 30×30 .

Because of the storage and speed problems encountered even when a high speed IBM 360/67 computer is used, future attention is to be focused on heuristic methods of solution.

2.3 Heuristic Methods

The heuristic methods adopted are based on roster characteristics; that is, size, capacity, or duration of the jobs in the roster. The heuristic rules can be shown to produce close to optimal results according to one measure of effectiveness, and there is strong indication that they are close to optimal according to the others. They are practical rules of thumb and are easier to apply than the method outlined above.

The rules developed schedule the jobs in a priority sequence determined from one or two of the three job characteristics: size, capacity, or duration. The primary scheduling rules do not define a unique schedule as they consider only one characteristic while the secondary rules break ties by considering one of the two remaining characteristics. The schedule is unique as the three characteristics are linearly dependent and two have been used.

Scheduling is done in a two-step procedure: the first step assigning the priority and the second step incorporating the shop

capacity constraint. The first step is a sorting procedure where the roster is rearranged in priority order with the highest priority job first and the lowest priority job last.

The second step consists of a routine which assigns the jobs in the roster from start to finish taking account the shop capacity constraint. Two types of routine are used; the first assigns the jobs in order of appearance, and the second searches ahead when necessary for a job which does not violate the capacity constraint and then returns to the point where it was originally stopped. A more detailed description of this operation is given in Section 4.2.2.C.III.

Reference to Section 1.5.2 shows that the primary scheduling rules assign priorities using size or duration only. A third possible priority rule based on capacity was not used since it was evident early in the study that the second of the two ways of performing step 2 produces far superior results (see Section 4.2.2). This is verified by the difference in performance of rules RANDOM and RANDOMFIT where the first uses method 1 on an unsorted roster, and the second uses method 2 on the same unsorted roster. (See Section 3.5.2.I, Table 14.)

The capacity based priority rule already arranges the jobs in capacity order defeating the purpose of the second method of assignment. Therefore, the capacity based priority system which defeats the use of the fitting procedure is eliminated.

However, a capacity sort was used in the tie breaking sort. Once all items of a set have one of three linearly dependent characteristics in common, it does not matter which of the other two characteristics is used to break ties as the schedule will be identical. The capacity sort was used because it is equivalent to and easier to program than a size sort with jobs of equal duration.

2.4 Examples of Scheduling Rules (Example Problem 2)

The four primary rules, RANDOM and RANDOMFIT are applied to a roster of five jobs scheduled through a shop with fixed capacity of four units. The roster is defined by capacity and duration vectors as follows:

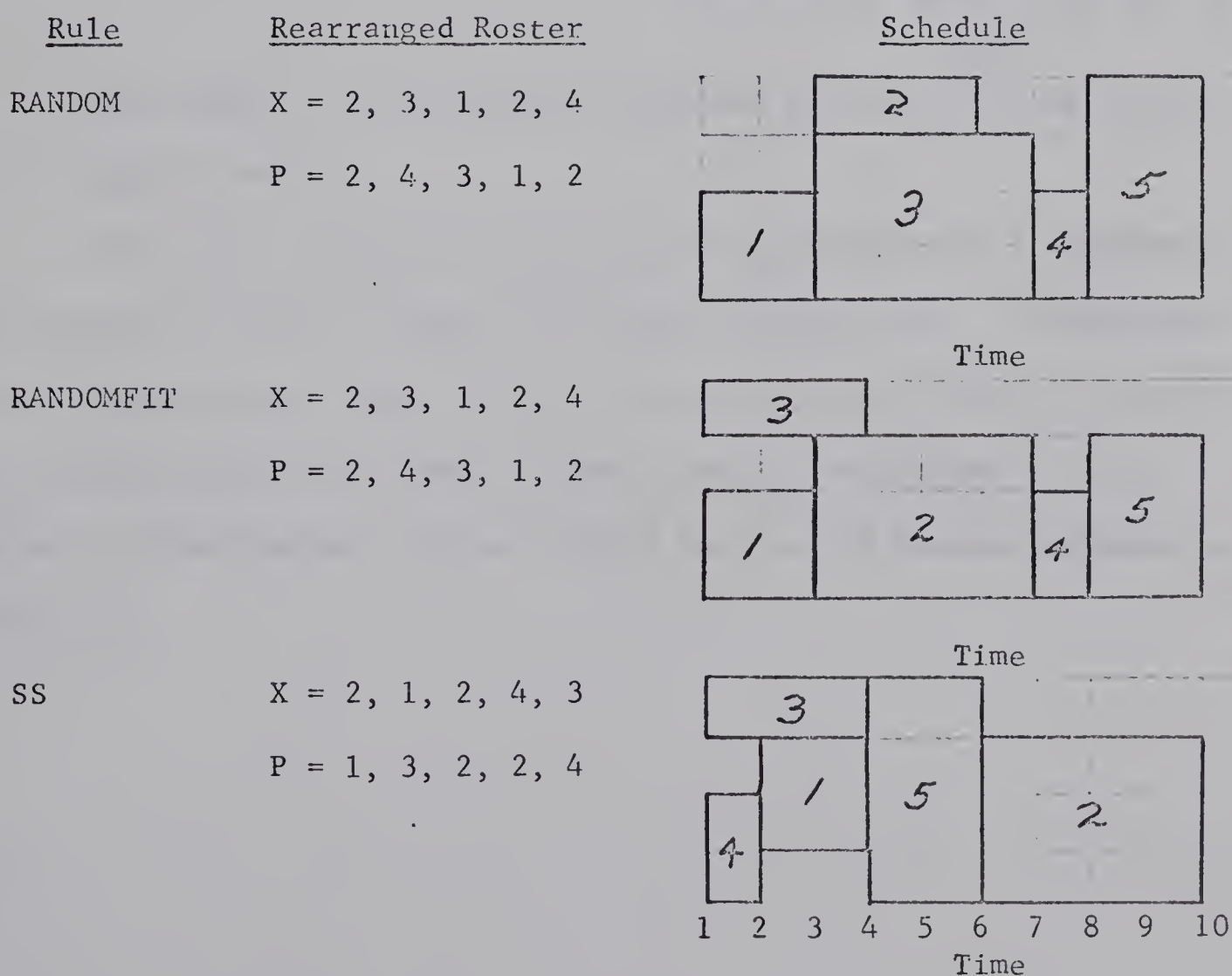
$$x = 2, 3, 1, 2, 4$$

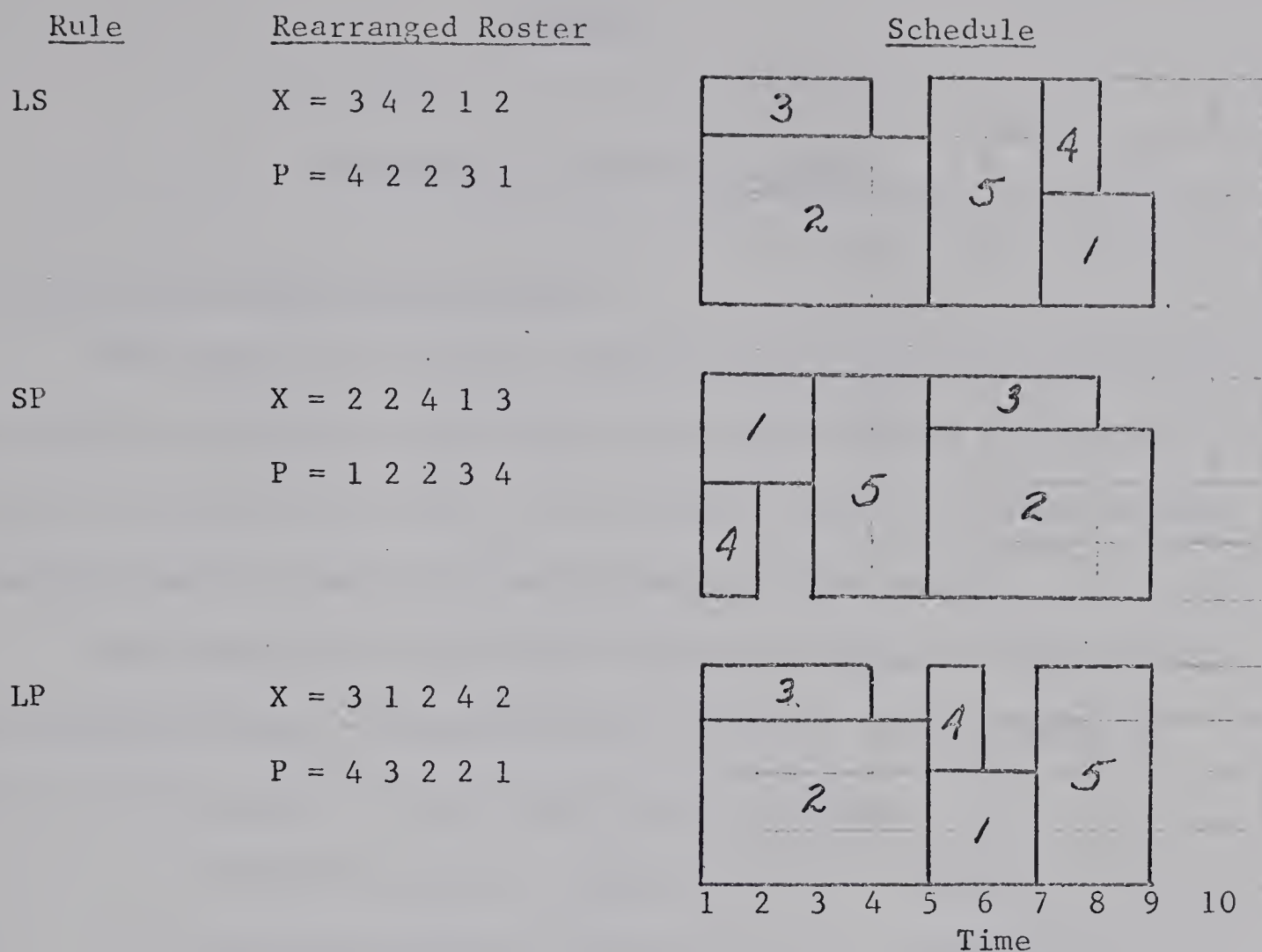
$$p = 2, 4, 3, 1, 2$$

Figure 5 shows the schedules produced and the order in which an attempt is made to schedule the jobs.

Figure 5

Comparison of Main Scheduling Rules Example 2





The numbers in the diagrams indicate the position the job had in the original roster.

From this example, a tendency for rules based on a smallest first priority to pile up jobs at the start can be seen. A combination of the fitting routine and a largest first discipline tends to intermix large and small jobs resulting in a more compact arrangement. The measures of effectiveness are calculated for each of these schedules in Section 3.3.

CHAPTER 3

EVALUATION OF SCHEDULING RULES

3.1 Data Used to Evaluate the Rules

The data used consists of sixty rosters containing twenty-five jobs each and a shop with nine units of capacity running as long as necessary to process the roster. For a given roster, the same process was used to generate both the capacity and duration vector.

The rosters were generated with two different processes and three different means, the result being the set of sixty rosters is broken into six groups of ten rosters each as follows:

- | | |
|--|-----------|
| 1. Uniform distribution range 1 to 5 | Symbol F3 |
| 2. Uniform distribution range 1 to 7 | Symbol F4 |
| 3. Uniform distribution range 1 to 9 | Symbol F5 |
| 4. Truncated Poisson distribution mean 3 | Symbol P3 |
| 5. Truncated Poisson distribution mean 4 | Symbol P4 |
| 6. Truncated Poisson distribution mean 5 | Symbol P5 |

The uniformly distributed rosters were generated using a library function (in the computer) which generates randomly distributed numbers within a specified range. To generate the Poisson rosters, the random distribution from this library function was used in a formula which transforms a random distribution into a Poisson distribution.

A chi square test is performed on each of the six sets of rosters to indicate to what degree the generated distribution approaches the intended distribution. The results of the test are calculated in Tables 6 and 7 and are summarized in Table 5 below.

TABLE 5

Chi Square Test Results on Rosters

Roster Type	X ² Capacities	X ² Durations	X ² Max.*
Uniform 1 to 5	.56	5.32	9.488
Uniform 1 to 7	9.85	11.00	12.592
Uniform 1 to 9	10.71	12.29	15.507
Poisson Mean 3	7.22	8.89	11.070
Poisson Mean 4	2.11	3.52	14.067
Poisson Mean 5	4.99	4.86	15.507

* (.05 level of significance)

Table 5 shows that the differences between the observed and theoretical distributions are in no case too large to be attributed to chance, and it is a reasonable assumption that they are generated by the intended distributions.

The Poisson distribution is modified by truncations at both ends. Values of zero are rejected since these represent a null job, and values greater than nine are rejected since these represent jobs of greater capacity than the shop can handle. Both truncations represent the real life situation as a job must have a greater-than-zero size, and a shop must occasionally reject a job that is beyond its capabilities.

Zero elements were eliminated by adding 1 to all elements preserving the shape of the distribution but shifting it horizontally or increasing the mean by 1. Elements of greater value than nine were eliminated by either regenerating the distribution when a larger element occurred or replacing the offending element by a random element from one to nine.

TABLE 6

Chi Square Calculation - Uniform Rosters

Observation	Capacities			Durations		
	Frequency	Expected	$\frac{(F-E)^2}{E}$	Frequency	Expected	$\frac{(F-E)^2}{E}$
<u>Range 1 to 5</u>						
1	54	50	16	63	50	169
2	49	50	1	53	50	9
3	47	50	9	46	50	16
4	51	50	1	44	50	36
5	49	50	<u>1</u>	44	50	<u>36</u>

$$X^2 = 28/50 = .56$$

$$X^2 = 266/50 = 5.32$$

X^2 max. 4 d.f. .05 level of significance = 9.488

Range 1 to 7

1	40	35.6	19.4	34	35.6	2.6
2	46	35.6	108.2	42	35.6	43.6
3	31	35.6	21.2	34	35.6	2.6
4	32	35.6	13.0	50	35.6	207.4
5	37	35.6	2.0	31	35.6	21.2
6	23	35.6	158.8	25	35.6	112.4
7	41	35.6	<u>29.2</u>	34	35.6	<u>2.6</u>

$$X^2 = 351.8/35.6 = 9.85$$

$$X^2 = 392.4/35.6 = 11.0$$

X^2 max. 6 d.f. .05 level of significance = 12.592

Range 1 to 9

1	28	28	0	29	28	1
2	40	28	144	20	28	64
3	21	28	49	31	28	9
4	28	28	0	35	28	9
5	31	28	9	40	28	144
6	26	28	4	20	28	64
7	31	28	9	34	28	36
8	19	28	81	24	28	16
9	26	28	<u>4</u>	27	28	<u>1</u>

$$X^2 = 300/28 = 10.71$$

$$X^2 = 344/28 = 12.29$$

X^2 max. 8 d.f. .05 level of significance = 15.507

TABLE 7

Chi Square Calculations - Poisson Rosters

Observation	Capacities			Durations		
	Frequency	Expected	$\frac{(F-E)^2}{E}$	Frequency	Expected	$\frac{(F-E)^2}{E}$
<u>Mean 3</u>						
1	36	34	.12	43	34	2.40
2	56	69	2.45	66	69	.13
3	74	69	.36	68	69	.01
4	45	46	.02	51	46	.54
5	30	22	3.04	11	22	5.50
6	7	9	1.23	9	9	.31
7	1	3		2	3	
8	1	1		0	1	

$$X^2 = 7.22$$

$$X^2 = 8.89$$

X^2 max. 5 d.f. .05 level of significance = 11.070

Mean 4

1	12	12	12	10	12	.33
2	42	37	.68	39	37	.11
3	59	57	.07	59	57	.07
4	49	57	1.12	53	57	.16
5	42	42		37	42	.60
6	26	25	.04	30	25	1.00
7	13	12	.08	15	12	.75
8	7	5	.12	3	5	.50
9	0	2		2	2	
9		1		1	1	

$$X^2 = 2.11$$

$$X^2 = 3.52$$

X^2 max. 7 d.f. .05 level of significance = 14.067

Mean 5

1	6	5	.20	3	5	.80
2	17	18	.06	18	18	
3	29	37	1.73	36	37	.03
4	55	49	.74	47	49	.08
5	52	49	.18	44	49	.51
6	39	39		48	39	2.07
7	24	26	.15	30	26	.62
8	12	15	.60	15	15	
9	16	7	1.33	9	7	.75
9		5			5	

$$X^2 = 4.99$$

$$X^2 = 4.86$$

X^2 max. 8 d.f. .05 level of significance = 15.507

TABLE 8

Comparison to Opposite Distribution

<u>Comparison</u>	<u>X² Capacities</u>	<u>X² Durations</u>	<u>X²* Max.</u>
Generated Poisson to Uniform	87.5	83.5	20.09
Generated Uniform to Poisson	132.3	79.5	20.09

* .01 level of significance

The differences between actual and presumed distributions are too great to be attributable to chance; and, therefore, we conclude that these hypothesized fits are not good.

It is, therefore, reasonable to state that the data generated approximates the intended distribution for each group. Each roster consists of twenty-five jobs drawn at random from these groups simulating the situation where the job characteristics follow some known distribution, and the shop receives a random sample from it.

It should also be noted that the truncation and shifting does not affect the Poisson distribution unduly. The truncations actually occurred rarely as for a Poisson distribution with a mean of 4, the theoretical distribution states that one element in 250 will exceed 9. Therefore, there is a .1 chance of one truncation occurring in each roster. For a mean of 5, there is a .5 chance of one truncation in any roster assuming only the durations will be truncated.

3.2 Criteria to Measure Effectiveness of Scheduling Rules

3.2.1 Definition of Measures of Effectiveness

Nine measures of effectiveness were developed to evaluate the fourteen scheduling rules described in Section 1.5.1. Some of these measures have been outlined in Section 1.5.2, and a more complete definition of them follows.

3.2.1.a Mean Start Time

This is the measure of effectiveness A described in Section 1.5.2 or $(+/NUM) \div PNUM$.

3.2.1.b Time Shop Occupied

This is the measure of effectiveness B of Section 1.5.2 or Γ/F .

3.2.1.c Average Number of Jobs in Process

The sum of the number of jobs in process each time period divided by the number of time periods the shop is occupied or $(+/JOB) \div (\Gamma/F)$.

3.2.1.d Overall Efficiency

The number of units of work required by the roster divided by the number of units of capacity occupied by it or $(+/(PxX)) \div (CC \times \Gamma/F)$.

3.2.1.e Mid-Range Efficiency

This is the measure of effectiveness C and the same as 3.2.1.d, except the second two quarters of the period of time the shop is occupied are considered instead of the whole period. The period considered is:

$$1 - SLACK \div CC \times 2 \times BEGIN$$

where $BEGIN = ((+/F) - (4/\Gamma/F)) \div 4$

and $SLACK = +/C [BEGIN + \frac{1}{2} \times 2 \times BEGIN]$

3.2.1.f Disutility Factor

This is the measure of effectiveness D and is defined as $+/PxXx (NUM-1)$.

3.2.1.g Sum of Start Times

Or $+/NUM$.

3.2.1.h Standard Deviation of Start Times

This is defined as $((+/(NUM-VECTMS)*2) \div \rho NUM)^*.5$

where $VECTMS = (\rho NUM) \rho ((+ / NUM) \div \rho NUM)$.

3.2.1.i Square Root of Variance of Start Times About Zero

This is similar to the standard deviation except the focal point is zero and not the mean of the distribution.

It is:

$$((+ / NUM*2) \div \rho NUM)^*.5.$$

The quantities in the above definitions are as defined in Section 1.1 except CC is C [1]. Three functions are defined in terms of these quantities and may be interpreted by reference to an APL manual.[1] They are defined as follows:

BEGIN - the lower quartile point of the shop occupied period taken to the nearest integer.

SLACK - the number of unused units of shop capacity between the first and third quartiles.

VECTMS - a vector of the same order as the roster vectors with elements equal to the mean start time.

In this notation, the operations are performed according to a right-to-left rule except where otherwise indicated by parentheses.

The basic symbols mean the following:

+/- - summation over the entire vector

r/ - maximum element in vector

ρ - number of elements in the vector

A*r - A raised to the power r

The more complex notation is explained in Section 4.1.3 and APL manuals.

3.2.2 Elimination of Measures of Effectiveness

Of the nine measures described in Section 3.2.1, only the four labelled A, B, C, and D were used in the evaluation of the scheduling rules. Two were eliminated because they were not relevant to the problem at hand, and three were eliminated because they were linearly related to others which were retained.

Measures 3.2.1.h and 3.2.1.j were intended to show how the start times are dispersed and point to schedules where one job is unduly delayed while the majority are processed quickly. In the extension of this study to a dynamic case as described in Section 1.3 where there is a flow of jobs into the roster as well as out, those measures would be relevant. They would indicate the scheduling techniques which allow a job to be kept away from the shop by the incoming stream of higher priority jobs. The measures are, therefore, included to give the model greater flexibility in spite of their lack of usefulness to the problem at hand.

Measures 3.2.1.a and 3.2.1.g are trivially related. They are $(+/NUM) \div \rho \text{ NUM}$ and $+/NUM$ where $\rho \text{ NUM}$ is the order of the roster vector, a constant. The mean start time was chosen as being more illustrative of the two.

A linear relationship also exists between the following measures:

$$3.2.1.b = B = \Gamma/F$$

$$3.2.1.c = B' = (+/JOB) \div (\Gamma/F)$$

$$3.2.1.d = B'' = (+/(P \times X)) \div (CC \times \Gamma/F)$$

To compare B and B'' note that in B'' P, X and CC are all constant given a roster and shop and are, therefore, independent of the

scheduling rule. Γ/F is the only factor dependent on the scheduling rule and, therefore:

$$B'' = \frac{k}{\Gamma/F} = \frac{k}{B} \quad \text{where } k = \frac{+/(P \times X)}{CC}$$

This is a linear inverse relationship between measures 3.2.1.b and 3.2.1.d. The constant is known.

To show that B and B' are related, we must show that for a given roster and shop $+/\text{JOB}$ is independent of the scheduling rule.

Each job runs for a number of units of time equal to its duration requirement and will occupy this time sooner or later in the schedule. No matter when it occupies the shop or how many other jobs are running, each job will eventually increment $\text{JOB } [N, N + 1, \dots, M]$ by 1 where N is its start time and M is its finish time. Therefore, $+/\text{JOB}$ is the sum of the durations of the jobs in the roster. This is a function of the roster and not the schedule and, therefore:

$$B' = \frac{k}{\Gamma/F} = \frac{k}{B} \quad \text{where } k = +/\text{JOB}$$

The number of units of time the shop is occupied was chosen as representative of these measures.

In summary, the four measures of effectiveness chosen as indicated in Section 1.5.2 are:

- A - 3.2.1.a Mean start time
- B - 3.2.1.b Time shop occupied
- C - 3.2.1.e Mid-range efficiency
- D - 3.2.1.f Weighted disutility factor

3.3 Application of the Measures of Effectiveness

For each of the sixty rosters, the nine measures of effectiveness were calculated and displayed by computer for each of the fourteen scheduling rules. As an example of this procedure, Example Problem 2 of Section 2.4 is considered. The four measures of effectiveness used are calculated for each of the six schedules in Table 10.

TABLE 10

Measures of Effectiveness - Problem 2

Common Quantities

$$+/(P_{XX}) = 4 + 12 + 3 + 2 + 8 = 29$$

$$CC = C [1] = 4$$

RANDOM Rule

$$NUM = 1, 3, 3, 7, 8$$

$$F = 2, 6, 5, 7, 9$$

$$BEGIN = 2$$

$$SLACK = 1$$

$$A = (1 + 3 + 3 + 7 + 8) \div 5 = 4.4$$

$$B = \text{Max. } [2, 6, 5, 7, 9] = 9$$

$$C = 1 - (1/(4 \times 2 \times 2)) = .94$$

$$D = (4 \times 0) + (12 \times 2) + (3 \times 2) + (2 \times 6) + (8 \times 7) = 98$$

RANDOMFIT Rule

$$NUM = 1, 3, 1, 7, 8$$

$$F = 2, 6, 3, 7, 9$$

$$BEGIN = 2$$

$$SLACK = 3$$

RANDOMFIT Rule (Cont'd.)

$$A = (1 + 3 + 1 + 7 + 8) \div 5 = 4.0$$

$$B = \text{Max. } [2, 6, 3, 7, 9] = 9$$

$$C = 1 - (3/(4 \times 2 \times 2)) = .81$$

$$D = (4 \times 0) + (12 \times 2) + (3 \times 0) + (2 \times 6) + (8 \times 7) = 92$$

SS Rule

$$\text{NUM} = 2, 6, 1, 1, 4$$

$$F = 3, 9, 3, 1, 5$$

$$\text{BEGIN} = 2$$

$$\text{SLACK} = 2$$

$$A = (2 + 6 + 1 + 1 + 4) \div 5 = 2.8$$

$$B = \text{Max. } [3, 9, 3, 1, 5] = 9$$

$$C = 1 - (2/(4 \times 2 \times 2)) = .88$$

$$D = (4 \times 1) + (12 \times 5) + (3 \times 0) + (2 \times 0) + (8 \times 3) = 98$$

LS Rule

$$\text{NUM} = 7, 1, 1, 7, 5$$

$$F = 8, 4, 3, 7, 6$$

$$\text{BEGIN} = 2$$

$$\text{SLACK} = 1$$

$$A = (7 + 1 + 1 + 7 + 5) \div 5 = 4.2$$

$$B = \text{Max. } [8, 4, 3, 7, 6] = 8$$

$$C = 1 - (1/(4 \times 2 \times 2)) = .94$$

$$D = (4 \times 6) + (12 \times 0) + (3 \times 0) + (2 \times 6) + (8 \times 4) = 68$$

SP Rule

$$\text{NUM} = 1, 5, 5, 1, 3$$

$$F = 2, 8, 7, 1, 4$$

SP Rule (Cont'd.)

$$\text{BEGIN} = 2$$

$$\text{SLACK} = 0$$

$$A = (1 + 5 + 5 + 1 + 3) \div 5 = 3.0$$

$$B = \text{Max. } [2, 8, 7, 1, 4] = 8$$

$$C = 1 - (0 / (4 \times 2 \times 2)) = 1.00$$

$$D = (4 \times 0) + (12 \times 4) + (3 \times 4) + (2 \times 0) + (8 \times 2) = 76$$

LP Rule

$$\text{NUM} = 5, 1, 1, 5, 7$$

$$F = 6, 4, 3, 5, 8$$

$$\text{BEGIN} = 2$$

$$\text{SLACK} = 3$$

$$A = (5 + 1 + 1 + 5 + 7) \div 5 = 3.8$$

$$B = \text{Max. } [6 + 4 + 3 + 5 + 8] = 8$$

$$C = 1 - (3 / (4 \times 2 \times 2)) = .81$$

$$D = (4 \times 4) + (12 \times 0) + (3 \times 0) + (2 \times 4) + (8 \times 6) = 72$$

3.4 Ranking of the Scheduling Rules3.4.1 Description of the Rankings

For each roster, the measures of effectiveness were calculated for each of the fourteen different schedules. Due to variations in particular rosters, the absolute values so calculated would vary considerably between rosters. As the object is to derive a scheduling rule and test its relative effectiveness regardless of the roster, a system of ranking the rules relative to each other was developed.

To illustrate this point, consider two rosters, A and B,

and two scheduling rules, 1 and 2. The efficiencies resulting from using the two rules on the rosters are shown in Table 11.

TABLE 11

<u>Scheduling Rule</u>	<u>Roster</u>	
	A	B
1	.9	.95
2	.88	.92

We may compare the performance of Rule 2 on roster B to rule 1 on roster A; observe that .92 is better than .9, and conclude scheduling rule 1 is better.

However, note that rule 2 performs better on the same roster, and the following method of ranking the rules illustrates this point. Assign a value 0 to the best rule for a given roster and a value 1 to the worst rule. Prorate numbers between 0 and 1 for the other rules depending on their relative position between the extremes as measured by the measure of effectiveness. Table 12 shows the application of this method to the absolute value rankings in Table 11.

TABLE 12

<u>Scheduling Rule</u>	<u>Roster</u>	
	A	B
1	0	0
2	1	1

This shows the consistent better performance of rule 1 relative to rule 2 and suppresses the difference in rosters. It

points out clearly the fact that we will get better results on any roster by using rule 1 rather than rule 2.

This ranking procedure was used to rank the calculated values of the measures of effectiveness obtained by application of the fourteen scheduling rules for each roster. The best performance was identified by a zero, and the worst performance by a 10,000. The intermediate values of the measures of effectiveness were assigned a value between 0 and 10,000 which was a linear function of the difference between them and the best and worst values.

3.4.2 Example of Ranking Procedure

The procedure described above is illustrated using the calculated values of the measures of effectiveness from the example of Table 9. Table 13 summarizes these results:

TABLE 13

Summary of Performance - Example Problem 2

<u>Scheduling Rule</u>	<u>Measure of Effectiveness</u>			
	A	B	C	D
Random	4.4*	9*	.94	98*
Randomfit	4.0 ^a	9*	.81*	92
Smallest Size	2.8**	9*	.88	98*
Largest Size	4.2	8**	.94	68**
Smallest Duration	3.0	8**	1.00**	76
Largest Duration	3.8	8**	.81*	72

* Worst

** Best

The table is considered column by column in order to compare all scheduling rules according to each measure of

effectiveness. The best performance in each column is indicated by ** and the worst by *.

Note that the best value for measures A, B, and D is a low number while for measure C, it is a high number. Consistency is obtained by assigning the best value 0 no matter whether it is high or low.

Table 13-A shows the rankings as they are calculated using this method. To illustrate the method, the rank for the element marked (α) on Table 12 is calculated:

$$10000 \times \frac{4.0 - 2.8}{4.4 - 2.8} = 7500$$

TABLE 13-A

Summary Rankings - Example Problem 2

<u>Scheduling Rule</u>	<u>Measure of Effectiveness</u>			
	A	B	C	D
Random	10000	10000	3333	10000
Randomfit	7500 α	10000	10000	8000
Smallest Size	0	10000	10000	10000
Largest Size	8750	0	3333	0
Smallest Duration	1250	0	0	2667
Largest Duration	6250	0	10000	1333

3.4.3 Application of the Ranking to the Scheduling Rules

Using the data from sixty rosters, a tabulation similar to Table 13 was made. The fourteen scheduling rules were evaluated according to each of the four measures of effectiveness and an average over the sixty rosters computed. The standard deviation was also calculated and used as explained in Section 3.5.

TABLE 14
MEAN AND STANDARD DEVIATION OF RANKINGS

1 Mean Start Time - Measure A

<u>Rule</u>	<u>Number</u>	<u>Mean</u>	<u>Std. Dev.</u>
RANDOM	60	8731.25	1583.83
RANDOMFIT	60	3313.43	863.42
SS	60	527.32	604.03
LS	60	8825.82	1382.28
SP	60	311.5	340.82
LP	60	6986.75	1409.24
SPSC	60	231.25	318.89
SPLC	60	443.45	439.89
LPSC	60	6603.23	1408.99
LPLC	60	7651.68	1379.98
SSSP	60	434.63	575.04
SSLP	60	636.35	578.44
LSSP	60	8847.75	1366.67
LSLP	60	8679.9	1489.56

2. Shop Occupied Time - Measure B

<u>Rule</u>	<u>Number</u>	<u>Mean</u>	<u>Std. Dev.</u>
RANDOM	60	9347.43	1250.51
RANDOMFIT	60	3554.4	2029.84
SS	60	6599.68	2385.07
LS	60	342.67	727.45
LS	60	4621.48	2550.19
LP	60	2244.93	2062.68
SPSC	60	4758.28	2317.07
SPLC	60	3962.85	2475.43
LPSC	60	3276.83	2395.51
LPLC	60	1411.72	1788.45
SSSP	60	6136.7	2256.92
SSLP	60	5684.9	3306
LSSP	60	242.32	624.19
LSLP	60	496.52	857.62

The average relative performance of the fourteen scheduling rules as measured by each measure of effectiveness presented in Table 14. From these, simple rules of thumb based on job characteristics for scheduling jobs through a job shop will be obtained. The standard deviations will be used to test the significance of hypothesized relationships.

In addition to Table 14, Table 15 shows the position ranking of each scheduling rule without regard for the varying differences between rules as shown in Table 14. This is useful in visualizing initial relationships and suggesting hypotheses for future testing.

TABLE 15

Rankings of Scheduling Rules

<u>Scheduling Rule</u>	<u>Measure of Effectiveness</u>			
	A	B	C	D
Random	12	14	14	14
Randomfit	7	7	8	6
SS	5	13	12	12
LS	13	2	2	2
SP	2	9	9	9
LP	9	5	5	5
SPSC	1	10	10	10
SPLC	4	8	7	7
LPSC	8	6	6	8
LPLC	10	4	4	4
SSSP	3	12	11	11
SSLP	6	11	13	13
LSSP	14	1	1	1
LSLP	11	3	3	3

3.4.4 Conclusions From the Rankings

Examining the data in Tables 14 and 15, several patterns become evident. These will be stated now and tested for significance in Section 3.5.

3.4.4a No matter what measure of effectiveness is used to evaluate it, the first-come-first-served rule is worst or nearly worst. Note that it is always worse than random fitting.

3.4.4.b Two different groups of scheduling rules perform markedly different from each other. One of the groups is based on priority given to a small quantity (size or duration) and the second is based on priority to a large quantity.

3.4.4.c Random fitting assignment ranks between the two groups according to all measures of effectiveness.

3.4.4.d The smallest first priority group of scheduling rules gives better mean start time while the largest first group gives better shop occupied time, efficiency, and weighted disutility factor.

The idea that fitting or searching ahead in the roster for a job which will utilize the unused capacity when the highest priority job exceeds it is good is confirmed by conclusion 3.4.4.e. If this relationship can be shown to be statistically significant, we are justified in omitting the capacity based priority rules as discussed in Section 2.3. The two rules RANDOM and RANDOMFIT were introduced to illustrate this point as the only difference between them is one uses the fitting procedure while the other does not.

We have chosen rules which behave in different ways and measures of effectiveness such that rules rank differently according to different measures. The fact that random fitting falls between the two groups of rules according to each measure of

effectiveness shows that we have chosen scheduling rules with opposite characteristics. We, therefore, appear to have a fairly comprehensive set of simple scheduling rules.

Table 15 is rearranged in order to illustrate relationships which occur as a function of the second or tie breaking priority assignment. Table 16 shows the same data as Table 15 except the two secondary scheduling rules of the same primary type are grouped around the primary scheduling rule.

TABLE 16

Rankings of Scheduling Rules

Scheduling Rule	Measure of Effectiveness			
	A	B	C	D
SSSP	3	12	11	11
SS	5	13	12	12
SSLP	6	11	13	13
SPSC	1	10	10	10
SP	2	9	9	9
SPLC	4	8	7	7
LSSP	14	1	1	1
LS	13	2	2	2
LSLP	11	3	3	3
LPSC	8	6	6	8
LP	9	5	5	5
LPLC	10	4	4	4

Except in one case, the behavior of the second sort is consistent. This is shown by each group of three numbers monotonically increasing or decreasing.

The conclusions regarding the secondary or tie breaking sort are presented in 3.4.4.e, 3.4.4.f, 3.4.4.g, and 3.4.4.h.

3.4.4.e If the primary scheduling rule is smallest size first, priority to smallest duration improves all measures A, B, C, and D.

3.4.4.f If the primary scheduling rule is smallest duration first, priority to smallest capacity improves measure A and priority to largest capacity improves measures B, C, and D.

3.4.4.g If the primary scheduling rule is largest size first, priority to largest duration improves measure A and priority to smallest duration improves measures B, C, and D.

3.4.4.h If the primary scheduling rule is largest duration first, priority to smallest capacity improves measure A and priority to largest capacity improves measures B, C, and D.

A second sort priority which behaves in a certain manner according to measure A appears to behave in the opposite manner according to measures B, C, and D.

Table 17 shows only the primary scheduling rules and their rankings as taken from Table 15 while Table 18 is a condensation of it using the ranking numbers 1 to 4 only. From these tables, the performance of the four primary groups of scheduling rules can be easily compared.

TABLE 17

Primary Scheduling Rule Rankings

Scheduling Rule	Measure of Effectiveness			
	A	B	C	D
SS	5	13	12	12
LS	13	2	2	2
SP	2	9	9	9
LP	9	5	5	5

TABLE 18

Primary Scheduling Rule Rankings

Scheduling Rule	<u>Measure of Effectiveness</u>			
	A	B	C	D
SS	2	4	4	4
LS	4	1	1	1
SP	1	3	3	3
LP	3	2	2	2

The main conclusions of the study are drawn from these tables. They are 3.4.4.j and 3.4.4.k.

3.4.4.j To obtain the least mean start time, give priority to the job with smallest duration. Ties are broken with priority to smallest capacity according to 3.4.4.f.

3.4.4.k To obtain the least time to complete all jobs, highest efficiency or minimum disutility factor, give priority to the job with largest size. Ties are broken with priority to smallest duration according to 3.4.4.g.

The relative value of other priority schemes is given by the rankings in Table 15.

There are apparent conflicts which may be statistically significant between the performance of a priority rule in the primary and secondary sort. As an example, priority to smallest duration results in better mean start times on the primary sort while on the tie breaking sort of jobs in size order the reverse is true.

Explanation of significant discrepancies is made in Section 3.5.3 following the statistical tests.

3.5 Comparison of the Scheduling Rules

3.5.1 t-Test for Significance of Difference of Means

A t-test for the difference of means is used to decide whether the observed differences in the scheduling rules is significant or due to chance. [6] Basic to this method is the assumption that we have drawn our sample from a normally distributed population of rankings.

If the entire population of rosters were formed in the same way, that is, had a set of jobs which would interlock into the same pattern, a unique rank as indicated by each measure of effectiveness would be observed (although different for each measure of effectiveness) for each scheduling rule regardless of the roster. The ranks are different for each roster because there are many chance variables affecting the makeup of each roster which in turn affects the interlocking pattern.

By the Central Limit Theorem, the cumulative effect of a multitude of random variables approaches normal distribution. In this case, the cumulative result of the variables affecting the interlocking pattern is the calculated value of the measure of effectiveness. It is, therefore, reasonable to conclude that there is a mean rank for each scheduling rule, and actual ranks for each individual roster are normally distributed about it with a fixed standard deviation. Our sample consists of sixty such actual ranks.

The t-test for difference between means* is a technique for deciding, given a sample from two populations, whether there is

* See reference [6] p. 233.

a difference in the population means. In this problem, the population means are the mean ranks assigned to the scheduling rules by a measure of effectiveness; and these are to be shown to be different by inference from the sample of sixty rosters. When this is done, we have established that one rule is better than another when we are trying to optimize a given measure of effectiveness. Accordingly, t-tests described in the above reference were performed on all possible combinations of two items from each column of Table 14.

3.5.2 Significance Tests on Hypothesized Conclusions

The relationships that were hypothesized from Tables 14 to 18 and presented in Section 3.4.4 are tested for statistical significance. All tests are performed at the 0.05 level of significance unless otherwise stated, and the number of degrees of freedom is 118. (Sixty items in each sample $n_1 + n_2 - 2 = 60 + 60 - 2 = 118$.)

3.5.2.a To test that the random first-come-first-served rule ranks last according to measures of effectiveness B, C, and D, its rank is compared to the rule which ranks second last according to each measure (see Section 3.4.4.a). Table 19 shows that there is a significant difference at the 0.01 level ($t > 2.617$) between this rule and the next worst rule according to all three measures of effectiveness.

TABLE 19

Comparison of RANDOM to Next Worst Rule

Measure	RANDOM Rank	Next Rule	Next Rank	t-Value
B	9347	SS	6599	7.83
C	8507	SSLP	7212	3.02
D	9735	SSLP	5982	10.78

The random first-come-first-served rule is not the worst rule according to measure of effectiveness A. However, Table 20 shows that the two rules which are worse than it are not significantly worse.

TABLE 20

Comparison of RANDOM to Worst Rules - Measure A

RANDOM Rank	Worse Rule	Worse Rule Rank	t-Value
8731	LS	8825	0.35
8731	LSSP	8847	0.43

The fact that the random first-come-first-served rule is the worst according to nearly all measures of effectiveness has therefore been established.

It has also been shown that the fitting procedure is necessary regardless of the measure of effectiveness in Table 21 by comparing RANDOM and RANDOMFIT and showing them to be different well beyond the 0.001 level of significance ($t > 3.373$).

TABLE 21

Comparison of RANDOM and RANDOMFIT Rules

<u>Measure of Effectiveness</u>	<u>RANDOM Rank</u>	<u>RANDOMFIT Rank</u>	<u>t-Value</u>
A	8731	3313	23.07
B	9347	3554	18.66
C	8507	3963	10.75
D	9735	2309	36.01

3.5.2.b The primary scheduling rules are significantly divided into two groups, one containing LP and LS, the other containing SP and SS. (See Section 3.4.4.b.) To show that the rules are significantly different in each group, the rule is chosen which ranks closest to the other group according to each measure of effectiveness. The ranks of the two rules thus chosen are compared in Table 22 which shows there is a difference between them at the 0.001 level of significance ($t > 3.373$).

TABLE 22

Test of Group Separation - Primary Rules Only

<u>Measure of Effectiveness</u>	<u>Shortest First Group</u>		<u>Largest First Group</u>		<u>t-Value</u>
	<u>Rule</u>	<u>Rank</u>	<u>Rule</u>	<u>Rank</u>	
A	SS	527	LP	6986	32.36
B	SP	4621	LP	2244	5.56
C	SP	4742	LP	2442	5.12
D	SP	3090	LP	1605	4.70

If the secondary rules based on these primary rules are included in their respective groups, an overlapping between

the groups occurs according to measures C and D. For measure A, there is still a strong separation while the separation according to measure B may still be due to chance. This is shown in Table 23.

TABLE 23

Test of Group Separation - All Rules Included

<u>Measure of Effectiveness</u>	<u>Shortest First Group</u>		<u>Longest First Group</u>		<u>t-Value</u>
	Rule	Rank	Rule	Rank	
A	SSSP	636	LPSC	6603	30.09
B	SPLC	3962	LPSC	3276	1.52
C	OVERLAP				
D	OVERLAP				

The primary rules are, therefore, shown to form into two groups regardless of the measure of effectiveness. The groups are on the basis of opposite performance according to any measure of effectiveness. By utilizing tie breaking priority rules, a degree of overlap can be created between the groups according to at least some measures of effectiveness.

3.5.2.c To show that the RANDOMFIT rule lies between the two groups of scheduling rules according to all measures of effectiveness, its rank is compared to that of the primary scheduling rule closest to each side of the intergroup boundary. (See Section 3.4.4.c.) These are the same rules that were compared in Table 22. As indicated in Table 24,

the RANDOM rule is significantly different from each group at the 0.05 level except for a marginal case indicated by (*).

TABLE 24

Location of the RANDOMFIT Rule - Primary Only

<u>Measure of Effectiveness</u>	<u>RANDOMFIT Rank</u>	<u>Lower Boundry</u>			<u>Upper Boundry</u>		
		Rule	Rank	t-Diff.	Rule	Rank	t-Value
A	3313	SS	527	20.30	LP	6986	17.07
B	3554	LP	2244	3.47	SP	4621	2.51
C	3963	LP	2442	3.43	SP	4742	1.65*
D	2309	LP	1605	2.73	SP	3090	2.49

* Significant at the 0.10 level

If the secondary rules are included in the groups, there is no significant separation for RANDOMFIT to fall into according to measures B, C, and D. However, according to measure A, it is separated from the lower and upper boundaries with greater than 99.9 percent confidence ($t > 3.373$) ($t = 19.78$ and 15.29 for upper and lower, respectively).

RANDOMFIT has, therefore, been shown to produce results midway between those produced by the two major groups in the set of primary scheduling rules according to all measures of effectiveness.

3.5.2.d This conclusion is elaborated in conclusions 3.4.4.e to 3.4.4.k and, therefore, will be tested in Sections 3.5.2.e to 3.5.2.k.

3.5.2.e The three scheduling rules in the SS (smallest size) priority group were examined to find the effect of the secondary sort. (See Section 3.4.4.e.) From Table 25 which shows the best and worst tie breaking priority according to each measure of effectiveness, we see that SP (shortest duration) is best according to all measures of effectiveness except B.

TABLE 25

Tie Breaking Effect - SS Priority

<u>Measure of Effectiveness</u>	<u>Best Tie Breaker</u>		<u>Worst Tie Breaker</u>		<u>t-Value</u>
	Rule	Rank	Rule	Rank	
A	SSSP	434	SSLP	636	1.90
B	SSLP	5684	SSSP	6136	0.87
C	SSSP	6630	SSLP	7212	1.20
D	SSSP	5262	SSLP	5982	1.55

The conclusion that SP priority tie breaking produces better results according to measures of effectiveness A, C, and D is illustrated by a weak relationship in that direction. The opposite effect is noted according to measure B although it is very weak.

3.5.2.f Table 26 compares the best and worst rule from the set of SP (smallest duration) based rules according to each measure of effectiveness. (See Section 3.4.4.f.)

TABLE 26

Tie Breaking Effect - SP Priority

<u>Measure of Effectiveness</u>	<u>Best Tie Breaker</u>		<u>Worst Tie Breaker</u>		<u>t-Value</u>
	Rule	Rank	Rule	Rank	
A	SPSC	231	SPLC	443	3.00
B	SPLC	3962	SPSC	4758	1.80*
C	SPLC	3748	SPSC	5530	3.77
D	SPLC	2339	SPSC	3809	4.17

Given that the primary priority is SP, this shows that there is a significant tendency at the 0.01 level ($t > 2.617$) for a smallest capacity priority to improve measure A and a largest capacity priority to improve measures C and D. There is also a strong indication that the latter is true for measure B as well (significant at 0.1 level).

3.5.2.g Table 27 below shows that the evidence supporting this conclusion is relatively weak except for measure of effectiveness B. (See Section 3.4.4.g.)

The rosters in the LS (largest size) priority groups are compared in the same way as before.

TABLE 27

Tie Breaking Effect - LS Priority

<u>Measure of Effectiveness</u>	<u>Best Tie Breaker</u>		<u>Worst Tie Breaker</u>		<u>t-Value</u>
	Rule	Rank	Rule	Rank	
A	LSLP	8679	LSSP	8847	.63
B	LSSP	242	LSLP	496	1.84
C	LSSP	488	LSLP	702	.95
D	LSSP	258	LSLP	367	.80

3.5.2.h As shown in Table 28 which considers the rules in the LP (longest duration) priority group, the conclusion of 3.4.4.h is verified very strongly past the 0.001 level of significance ($t > 3.373$).

TABLE 28

Tie Breaking Effect - LP Priority

<u>Measure of Effectiveness</u>	<u>Best Tie Breaker</u>		<u>Worst Tie Breaker</u>		<u>t-Value</u>
	Rule	Rank	Rule	Rank	
A	LPSC	6603	LPLC	7651	4.08
B	LPLC	1411	LPSC	3276	4.79
C	LPLC	1545	LPSC	3520	5.27
D	LPLC	1021	LPLC	2431	4.60

3.5.2.j To show that the SPSC rule gives the best result according to measure of effectiveness A, we compare SP to each of the other three primary rules to confirm that it is significantly better than all of them. (See Section 3.4.4.j.) The secondary rule is verified by conclusion 3.5.2.f which states that if the SP rule is used, measure A is improved by giving priority to smallest capacity (SC). Table 29 shows that SP is the best primary rule at the 0.05 level of significance ($t > 1.98$).

TABLE 29

Comparison of SP to Primary Rules - Measure A

<u>SP Rank</u>	<u>Other Rule</u>		<u>t-Value</u>
	Rule	Rank	
311	SS	527	2.39
	LS	8825	45.93
	LP	6986	35.36

3.5.2.k To show that the LSSP rule gives the best result according to measures of effectiveness B, C, and D, we compare LS to each of the other three primary rules to confirm that it is better than all of them. (See Section 3.4.4.k.) Conclusion 3.5.2.g states that if the LS rule is used on the primary priority, there is a strong indication that priority should be given to smallest duration to give the best result according to measures of effectiveness B, C, and D.

Table 30 compares LS to each of the other primary scheduling rules for each measure of effectiveness and shows that this rule is superior to the other primary rules at the 0.001 level of significance ($t > 3.373$).

TABLE 30

<u>Measure of Effectiveness</u>	<u>LS Rank</u>	<u>Other Rule</u>		<u>t-Value</u>
		<u>Rule</u>	<u>Rank</u>	
B	342	SS	6599	19.27
		SP	4621	12.39
		LP	2244	6.68
C	700	SS	6833	15.06
		SP	4742	10.61
		LP	2442	5.07
D	354	SS	5660	15.18
		SP	3090	9.76
		LP	1605	5.79

Therefore, a one best primary rule has been isolated and found to be significantly different from all the others according to each measure of effectiveness. A secondary rule (SPSC) which produces a unique schedule has been shown to produce a significantly best result according to measure A. There is strong indication that the unique secondary rule LSSP produces the best result according to measures B, C, and D. While the results of Table 27 which analyzes the second sort effect when the primary priority is largest size (LS) are not significant, they are consistent with each other and with other results which have been verified more strongly.

Table 18 of Section 3.4.4 shows the relative rankings of each primary scheduling rule according to each measure of effectiveness. To show that there is a significant difference between all primary rules, Table 31 shows the t-value of the difference between each rule and the next worst rule.

TABLE 31

Significance Test on Table 18

Scheduling Rule	<u>Measure of Effectiveness</u>			
	A	B	C	D
SS	32.36	WORST RULE	WORST RULE	WORST RULE
LS	WORST RULE	6.68	5.07	5.79
SP	2.39*	4.35	4.19	6.14
LP	7.16	5.56	5.12	4.70

Therefore, the separation of the primary scheduling rules is very strong as it is significant at the 0.001

level ($t \geq 3.373$) except for one case (*) which is significant at the 0.01 level of significance.

3.5.3 Analysis of General Results

It seems reasonable that giving priority to the smallest jobs should minimize mean start time. However, there is a significant difference in the result depending on whether "smallest" means smallest size or smallest duration. On the tie breaking sort, it is noted that either smallest or largest duration first gives the best result according to measure A depending on whether the primary rule is smallest or largest size first, respectively. The reason for this shift is not immediately evident. When the primary rule is based on duration rather than size, a tie breaking priority to the smallest size or capacity job gives the best result according to measure A regardless of whether the primary rule is largest or smallest duration first.

Measures B, C, and D all give the same general rankings to the scheduling rules. It is reasonable that an action which improves the time the shop is occupied will also improve efficiency, but it is not as evident why it should improve the weighted disutility or profit-loss factor.

For measure D, the weighted disutility factor, it is advantageous to complete as many large jobs as soon as possible. To do this, one of the rules in the "largest first" group is used and is shown to be superior to any in the "smallest first" group. In the tie breaking sort when the primary rule is largest duration, priority to largest capacity further improves measure D. However, when the primary rule is largest size, a seemingly opposite effect

occurs as priority to smallest duration improves measure D. Note that largest duration first is better than smallest duration first in the primary set of rules.

For measures B and C, shop occupied period and mid-range efficiency, a largest duration or size priority is also best. It is not immediately evident why this should be so as the same result should occur whether the schedule or its mirror image is considered. In addition, the same conflict between primary and secondary scheduling rules exists as does for measure D.

In summary, three points must be explained or rationalized:

1. Why do the "largest first" rules optimize measures B and C, shop occupied time and mid-range efficiency.
2. Why is measure A optimized by a sort according to duration while the other three measures B, C, and D are optimized by a sort according to size.
3. Is there a real conflict between the behavior of the primary and secondary rules and what is the reason for it.

To facilitate the analysis, a full display of the schedule, number of jobs in process, and amount of capacity utilized in each time period as well as the summary results was made for five rosters, one from each group. An example of this display is Table 32 made for a roster of ten jobs.

TABLE 32

SAMPLE COMPUTER OUTPUT
APPLICATION OF LSSP RULE TO ROSTER OF TEN JOBS

CAPACITIES, PROCESS TIMES AND NUMBER OF JOBS IN ROSTER

8	4	6	2	7	3	9	2	5	7
5	8	2	5	9	2	4	9	5	1
10									

REARRANGED ROSTER IN ORDER SCHEDULING IS ATTEMPTED

CAPACITIES	7	8	9	4	5	2	6	2	7	3
PROCESS TIMES	9	5	4	8	5	9	2	5	1	2

SCHEDULE IS AS FOLLOWS

PROCESS TIME	CAPACITY	START	FINISH
9	7	1	9
9	2	1	9
5	8	10	14
4	9	15	18
8	4	19	26
5	5	19	23
5	2	24	28
2	3	24	25
2	6	27	28
1	7	29	29

JOBS IN PROCESS	2	2	2	2	2	2	2	2	2	2	1	1
1	1	1	1	1	1	1	2	2	2	2	3	3
2	2	2	1									
UNUSED CAPACITY	0	0	0	0	0	0	0	0	0	0	1	1
1	1	1	0	0	0	0	0	0	0	0	0	0
3	1	1	2									

SUMMARY CHARACTERISTICS

16.9	29	1.724137931	0.9540229885	0.9603174603
2766	169	9.565040512	19.4190628	

SUMMARY CHARACTERISTICS ARE:

Mean Start Time

Shop Occupied Time

Average Number of Jobs in Process

Total Efficiency

Mid-Range Efficiency

Disutility Factor

Sum of Start Times

Standard Deviation of Start Times

Square Root of Variance About O Start Time

To explain why a largest duration or size first type of priority rule is best according to measures of effectiveness B and C, we examine the number of jobs running in each time period in the cycle.

FIGURE 5
Uniform - Mean 3

Number of Jobs Running

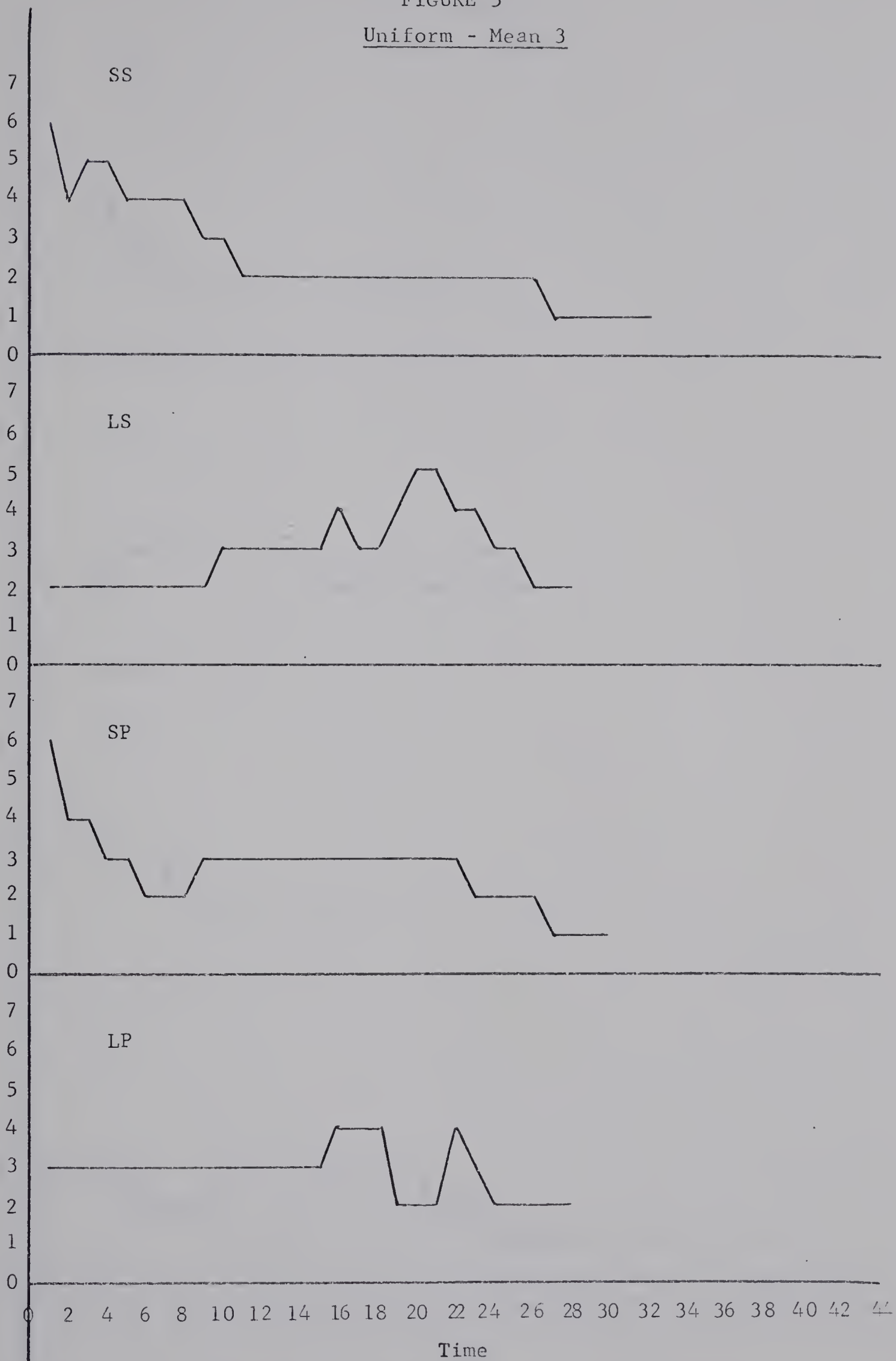


FIGURE 6

Poisson - Mean 3

Number of Jobs Running

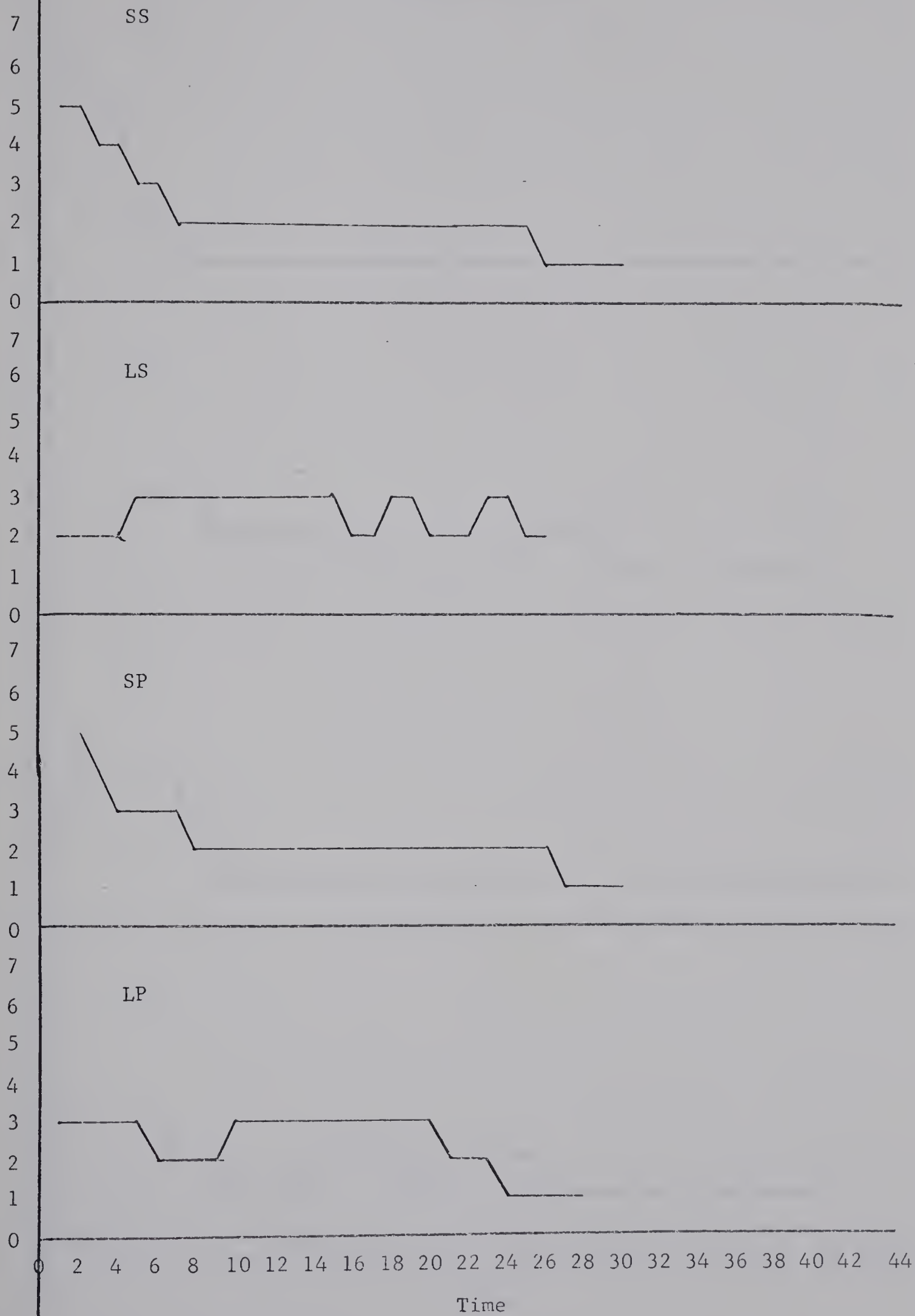


FIGURE 7

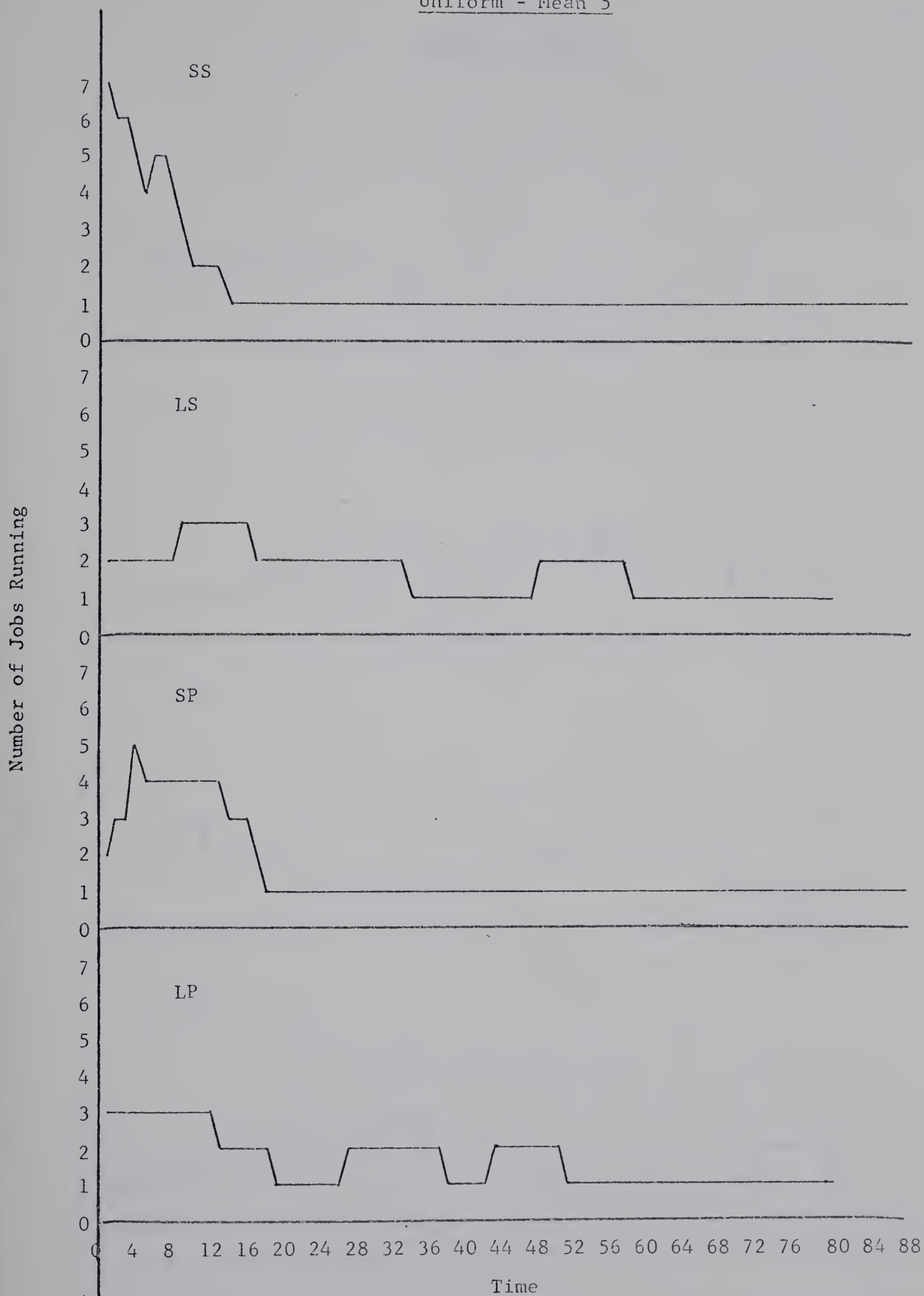
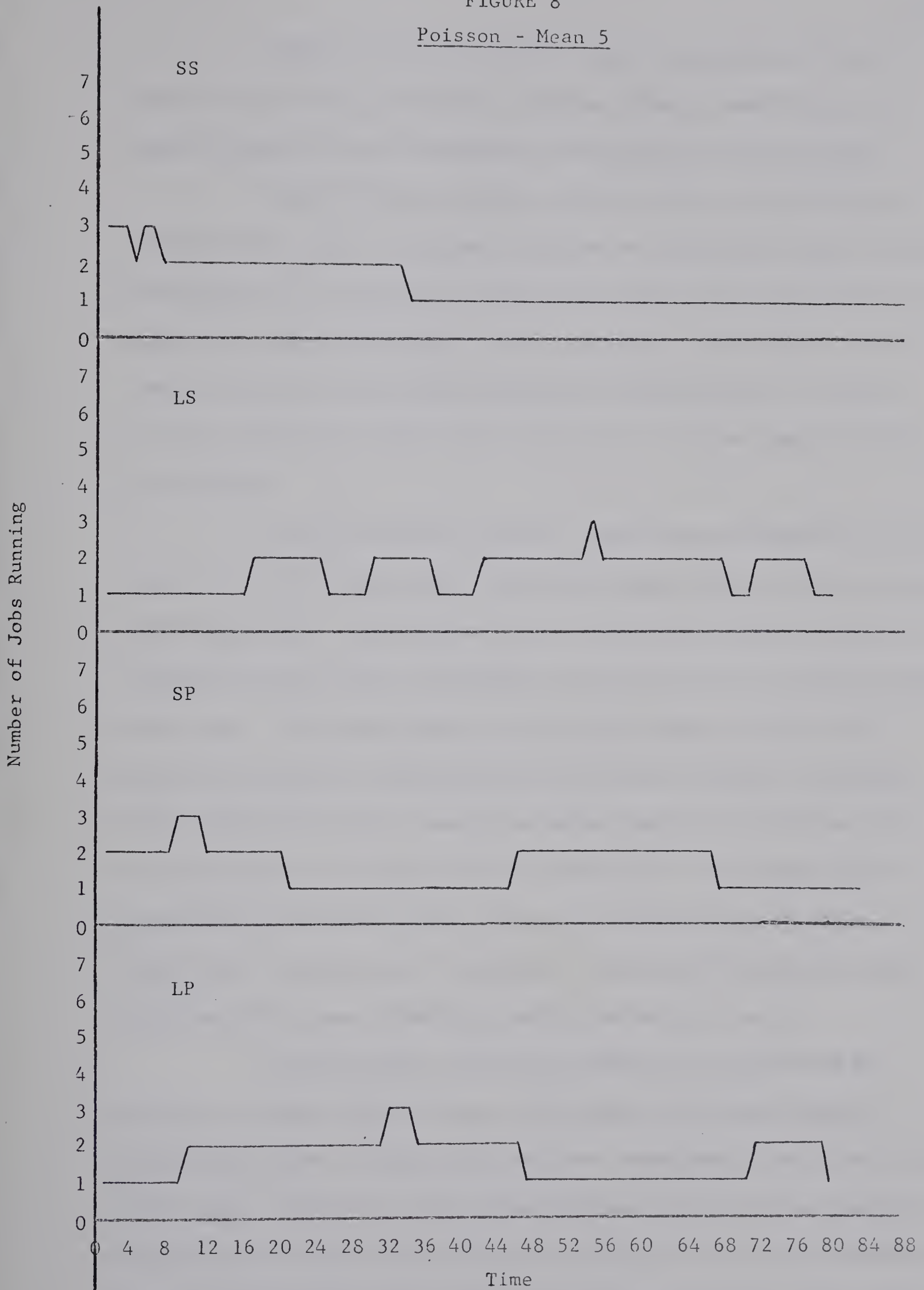
Uniform - Mean 5

FIGURE 8

Poisson - Mean 5

Figures 5 to 8, inclusive, show the number of jobs in process in each time period for the four primary scheduling rules using a typical roster from each of the groups of roster types.

The difference between the plots for smallest duration or size first rules and largest duration or size first rules is that the former have a pronounced peak at the start while the latter are relatively uniform throughout the time period. This means that a "shortest first" rule tends to bunch up a large number of jobs at the start while the other type of rule runs the same number of jobs at all times.

The jobs which a shortest first routine schedule at the start are all the small jobs leaving the large jobs to be run at the end of the cycle. The largest first routine will give priority to a large job when space is available rather than fill it with several small jobs. Often more than one large job cannot be run at the same time because of the capacity restriction; and when a smallest first scheduling rule is used, the large jobs are trailed out one at a time after the small jobs are scheduled. The largest first routine will schedule a small job in the unused capacity while a large job is running; and, therefore, the amount of shop time which would have been used scheduling small jobs only is saved.

The mid-range efficiency, measure C, is improved by priority to large jobs as these rules permit the unused capacity which would occur if large jobs only are scheduled to be filled with small jobs. Therefore, the reason a largest first routine improves both measures B and C is because it is able to fit the jobs together better by utilizing small jobs to fill the gaps between large ones.

Measure of effectiveness A has been shown to be optimized by giving priority to smallest duration rather than smallest size. If the primary rule was priority to smallest size, a longer than necessary duration is introduced at the start of the schedule as some small size jobs may have long durations. The result is an advancement of the start and finish times of all jobs subsequently scheduled. This means that the mean start time will be longer than if the smallest duration jobs were scheduled first.

Measures of effectiveness B, C, and D require that the largest size rather than largest duration job be given priority on the primary sort. As we have seen, a good fit is essential to good performance according to time shop is occupied and mid-range efficiency (measure B and C). It must be then that a size rather than duration based routine produces a better fit.

For a good fit, there must be a great variety of jobs available at each decision point or point at which a new job may be scheduled. Since both capacities and durations are randomly generated and size is the product of the two, there will be a greater variety of capacities in an equal size group than in an equal duration group. This is because there are two random variables affecting the size group while only one affects the duration group.

As an example of why a good mix is important to good fitting, consider what may happen when a three-unit capacity bloc becomes available in the first part of the cycle and there are three jobs, one requiring two units and two requiring one unit of capacity. If the two-unit job occurs first, it is scheduled; and the one-unit jobs are easily scheduled later. If both one-unit jobs occur ahead

of the two-unit job, they are both scheduled leaving a more difficult job to schedule later. If they occur mixed, it is possible to schedule the two-unit job immediately as well as one one-unit job.

The largest size first scheduling rule, therefore, results in a more varied roster making a good fitting pattern more likely than if one parameter were monotone increasing or decreasing throughout.

According to measures of effectiveness B, C, and D, largest duration first is a better rule than shortest duration first on the primary sort while to tie break it is better to choose smallest duration first after a primary sort according to size. This is equivalent to giving priority to the largest capacity job on the secondary sort. By doing this, if a large pocket of unused capacity occurs at the start of the schedule, it is filled immediately by a large capacity job. If it was filled by a small capacity job, this would leave the other larger job until later where only small capacity vacancies may occur. The fit would be poorer as these vacancies would be left unoccupied because the small jobs were scheduled, and the larger job would run by itself at the end.

3.6 Unusual Rosters

3.6.1 Classification of Unusual Rosters

An unusual roster is one which consistently produces unusual results according to any or all of the measures of effectiveness. The ranks of each scheduling rule according to each measure of effectiveness for the sixty rosters are considered and the mean and standard deviation calculated. From this, a 95 percent

confidence interval can be calculated; and an unusual result is a rank that falls outside this confidence interval.

Table 33 shows the frequency of occurrence of unusual ranks for each roster according to each measure of effectiveness, and Table 34 is a composite for all measures of effectiveness. Both horizontal and vertical scales in the tables are used to identify the rosters where the horizontal scale identifies the roster type as follows:

- 10 - Uniform roster range 1 to 5
- 20 - Uniform roster range 1 to 7
- 30 - Uniform roster range 1 to 9
- 40 - Poisson roster mean 3
- 50 - Poisson roster mean 4
- 60 - Poisson roster mean 5

The numbers on the vertical scale represent the individual roster in each of the groups above. For example, the code number for the roster represented by column 40 and row 5 is 45 and is a Poisson roster with mean 3. As it is in the sixth row, it is the sixth in the matrices containing Poisson rosters with mean 3. By referring to Section 4.3.3, the location of the capacities for this roster is ROSCP3 [6;] and the durations are located in the matrix ROSDP3[6;].

TABLE 33

Frequency of Unusual Rosters

	<u>Average Start Time</u>					
	10	20	30	40	50	60
0	2		2			5
1	5			2		1
2				1	2	
3	2	1			2	
4						
5						
6						
7				1		
8		2				4
9		1		2	1	6

<u>Shop Occupied Time</u>						
0	1			1	1	
1			5	1		
2				1	1	1
3	2	1	2			
4	1	2		1		
5	2		2	2		
6						
7		2		1		
8			4		2	2
9		1			1	

<u>Average Efficiency</u>						
0	1	3				
1		1	5	2		
2						2
3	1		2		1	5
4	2	1		1		
5	1				1	
6		1				
7		3		1		
8			1			2
9	1	3		1		

<u>Disutility Factor</u>						
0		1				1
1			3			
2				1		1
3		1		1		2
4	2	3				
5			2	1		
6			2			
7		2	2	1		
8			3			7
9		1		1		

TABLE 34
Composite of Table 33

	10	20	30	40	50	60
0	4	4	2	1	1	6
1	5	1	13	5		1
2				3	3	4
3	5	3	4	1	2	8
4	5	6		2		
5	3		4	3	1	
6		1	2			
7		7	2	4		
8		2	8		2	15
9	1	6		3	3	6
Total	23	30	35	22	13	40

The maximum element in any cell in Table 33 is 14 which would occur if every scheduling rule applied to the roster gave an unusual result according to the measure of effectiveness. If the technique has isolated 5 percent of the cases as deviant (corresponding to 95 percent confidence interval), there should be 5 percent of $14 \times 4 \times 60$ or 168 observations in Table 34. There are actually 163 observations.

Table 35 shows the number of rosters with each frequency of unusual results. Note that the average roster would produce 2.7 unusual results according to any of the four measures of effectiveness.

TABLE 35

Number of Rosters and Frequency of Unusual Results

<u>Frequency</u>	<u>Number</u>
0	19
1	8
2	6
3	8
4	6
5	4
6	4
7	1
8	2
9	
10	
11	
12	
13	1
14	
15	<u>1</u>

60 Rosters

There are five rosters whose performance could be considered unusual; of these, three are moderately unusual, and two are extremely unusual. The matrix location codes (see Section 4.3.3, Table 47) for the moderately and extremely unusual rosters are shown in Table 36.

TABLE 36

Location of Unusual Rosters

	<u>Code</u>	<u>Matrix Location</u>	<u>Description</u>
Moderate	27	ROSF4 [8;]	Uniform Range 1-7
	38	ROSF5 [9;]	Uniform Range 1-9
	63	ROSP5 [4;]	Poisson Mean 5
Extreme	31	ROSF5 [2;]	Uniform Range 1-9
	68	ROSP5 [9;]	Poisson Mean 5

The last two rosters were analyzed to try and find a simple reason for their unusual performance.

Another roster which is of interest is code 60 or ROSP5 [1;]. This roster has the greatest number of unusual occurrences of any roster according to measure A and none according to any other measure of effectiveness.

3.6.2 Analysis of Unusual Rosters

Referring to Table 36, it may appear that a roster with large average size relative to the size of the shop is prone to unusual results. However, Table 27 shows that seven out of twenty or 35 percent of the largest rosters (code 30 and 60) have no unusual occurrences at all, while nineteen in sixty or 32.5 percent of the whole sample have no unusual occurrences. It, therefore, seems as likely that a large roster will have no unusual occurrences as an average sized one.

The column totals of Table 34 show the frequency of unusual occurrences for each roster class. By using a chi square test for goodness of fit, we compare these to a uniform distribution of unusual occurrences to see if there is any significant relationship between roster size or type and the frequency of unusual occurrences. A chi square value is calculated as $X^2 = 14.3$; and, therefore, the hypothesis that a uniform distribution applies to the data is rejected at the 0.05 level of significance ($X^2_5 \text{ max} = 11.07$). However, when the two extremely unusual rosters are deleted from consideration, $X^2 = 8.2$; and the fit is acceptable at the 5 percent level of significance.

Therefore, the proneness to unusual results does not seem

to be associated with size of the jobs in the roster. If a simple causal factor is to be found, it must be related to the distribution of jobs in the roster; e.g., dispersion of jobs, the way capacities and durations are related, etc.

The unusual uniformly distributed roster is considered first. It has been drawn out of a population which has been shown in Table 6, Section 3.1, to follow a uniform distribution with range 1 to 9. The unusual roster is compared in the same manner to a theoretical uniform distribution, and X^2 values of 9.90 and 8.40 are calculated for capacity and duration, respectively. At the 5 percent level of significances (X^2 max = 15.507), we cannot reject the hypothesis that this roster forms a uniform distribution. In fact, it is somewhat closer to the theoretical distribution than the population from which it is drawn. Therefore, no unusual distribution of the roster has been found.

The capacities and durations may associate in an unusual manner; i.e., all large capacities with large durations, etc. Table 37 shows the capacities, durations, and sizes of jobs in this roster.

TABLE 37

Capacities, Durations and Sizes - Roster ROSF5 [2:]

Capacity	6	2	8	2	9	3	3	1	2	6	7	8
Duration	6	3	7	2	2	5	8	8	6	7	3	2
Size	36	6	56	4	18	15	24	8	12	42	21	16

Capacity	7	7	7	6	1	6	3	3	7	2	7	5	5
Duration	5	3	6	5	4	3	4	4	5	5	4	3	1
Size	35	21	42	30	4	18	12	12	35	10	28	15	5

The number of times a large or small capacity has associated with a large or small duration has been tabulated in Table 38 where "large" and "small" denote falling above and below the mean, respectively.

TABLE 38

Capacity - Duration Association - Roster ROSF5 [2:]

<u>Duration</u>	<u>Capacity</u>	
	Large	Small
Large	6	4
Small	8	7

This shows a slight tendency to have a greater number of large capacity and small duration jobs although the effect is not pronounced. There is also a tendency for the jobs to be smaller than normal, as eight exceed the normal size of 25 while seventeen jobs are less than it.

The standard deviations of the capacities and durations are a measure of their dispersion. The capacities of the roster in question have a standard deviation of 2.41 compared to an average for the set of ten of 2.53. It is fourth from the lowest. The durations have the lowest standard deviation, 1.88, compared to an average of 2.47. This is somewhat low, but it may be noted that the next lowest duration standard deviation roster has only six unusual rankings.

Table 39 shows the scheduling rules for which this roster had unusual rankings. There may be a complex mix of jobs in the roster arranged in such a way as to make one scheduling rule

extremely useful or useless distorting the results according to any one measure of effectiveness. In Table 39, the first number indicates the rank of the roster in question; and the second is the average rank for all rosters.

TABLE 39

Comparison of Unusual Results of ROSF5 [2:] to Normal

<u>Scheduling Rule</u>	<u>Measure of Effectiveness</u>			
	A	B	C	D
RANDOM				
RANDOMFIT				
SS				
LS		2143/342	3604/700	
SP				
LP		8571/2244	9352/2442	7300/1605
SPSC				
SPLC				
LPSC		8571/3276	9352/3250	7551/2431
LPLC		8571/1411	9352/1545	7139/1021
SSSP				
SSLP				
LSSP				
LSLP		2143/496	3604/702	

The fact that only one class of scheduling rules is subject to unusual occurrences suggests that the unusual occurrences are not due to random placement of jobs in the roster which would make a random rule very good or bad distorting the rest of the results.

A characteristic of this roster interacts with the LP scheduling priorities producing much worse than usual results according to measures of effectiveness B, C, and D.

These are the measures which require that the jobs be fit well together to optimize. Since there seems to be no unusual roster characteristics and there are sufficient small jobs to make the fitting procedure work, we must conclude that this unusual result cannot be attributed to prominent, simple roster characteristics. It is due to a complex interaction of jobs which affects their ability to nest into the shop without leaving large blocs of unused capacity.

The other extreme roster is a Poisson distributed roster with mean 5 and is drawn from a population which has been shown (Table 7, Section 3.1) to have these characteristics. A chi square comparison of this roster frequencies to theoretical frequencies for this distribution is given in Table 40.

TABLE 40

Frequency Comparison - Roster ROSP5 [9:]

<u>Observation</u>	<u>Capacity</u>			<u>Duration</u>		
	Frequency	Expected	$\frac{(F-E)^2}{E}$	Frequency	Expected	$\frac{(F-E)^2}{E}$
1	0	.5	.50	1	.5	.50
2	0	1.8	1.80	0	1.8	1.80
3	0	3.7	3.70	3	3.7	.13
4	5	4.9	0.00	1	4.9	3.10
5	8	4.9	1.96	12	4.9	10.25
6	5	3.9	0.31	5	3.9	.31
7	5	2.6	2.22	2	2.6	.14
8	0	1.5	1.50	1	1.5	.17
9	2	.7	.53	0	.7	1.20
9	0	.5	_____	0	.5	_____

$$\chi^2 = 12.52$$

$$\chi^2 = 17.70$$

χ^2 limit for 8 degrees at freedom and .05 level of significance = 15.507

This roster does not follow the Poisson distribution as well as the average as X^2 is high and, in fact, too high for the durations. Examination of the roster shows it is more tightly packed around the mean than is the Poisson distribution. The standard deviation of the capacities is the second lowest in the group; and for the durations, it is the lowest. The chi square test confirms that the tendency of the durations to be 5 or 6 is significantly different than that which would be expected from a Poisson distribution.

Table 41 is similar to Table 39 and illustrates the scheduling rules which produce the unusual ranks. Again, the first number indicates the rank of ROSP5 [9:] and the second number is the average rank for all rosters.

TABLE 41

Comparison of Unusual Results from ROSP5 [9:] to Normal

<u>Scheduling Rule</u>	<u>Measure of Effectiveness</u>			
	A	B	C	D
RANDOM				7392/9735
RANDOMFIT				
SS	2266/527			
LS				5962/354
SP				
LP			8239/2242	6201/1605
SPSC				
SPLC				0/2339
LPSC	9453/6603	10000/3276	10000/3250	10000/2431
LPLC		5455/1411		
SSSP	2277/434			
SSLP	2344/636			
LSSP				5835/258
LSLP				3790/367

From this table, a general tendency to produce poorer results is noted for any scheduling rule type measured by any measure of effectiveness. There is a strong tendency for a random rule to be better than the usual best sorting and fitting rule; and for measure of effectiveness D, four fitting rules, SS, LPSC, SSSP, and SSLP, are worse than random first-come-first-served.

For a typical roster, the random first-come-first-served rule is the worst; and the other rules which incorporate the fitting procedure show a marked degree of improvement over it. In the roster, the jobs must be arranged in such a way as to permit random first-come-first-served to produce nearly as good a fit as a fitting routine; in other words, the jobs are already in nearly the order in which a fitting routine would assign them. Therefore, the other rules do not make much of an improvement over first-come-first-served; and their ranks show considerably worse than average.

Also, the very tight compaction in this roster means the jobs are tending toward being identical. In the extreme case where the jobs are identical, it would not matter which rule was used as the schedules would all be the same.

The roster ROSP5 [10:] is of interest because all unusual occurrences were indicated by measure A and none were indicated by the other measures. While SP remains the best scheduling rule for this roster according to measure A as is the case in the general result, the results with the LS group are improved; and with the SS group, they are worse.

Again in this roster, there is no unusual pattern to the manner in which capacities and durations are associated, the

distribution of the capacities, and durations or the size of the jobs in the roster. There appears to be no simple explanation for the unusual behavior of this roster as indicated by measure A.

The above approach has been to isolate an unusual roster and try to find some characteristic which causes the unusual behavior. Except for deciding that it does not matter what rule is applied to a roster of identical jobs, the effort is unsuccessful and the deviations, therefore, can only be attributed to chance arrangements of the jobs in the roster. A practical method of predicting what roster will behave unusually and how it will behave before it is scheduled has not been found.

The apposite approach is now taken, that is, isolate a possible factor which may cause unusual results and see if it in fact does. The most promising causal factor appears to be standard deviation of the items in the roster. The Poisson and uniform rosters with highest and lowest standard deviations are examined to see if they result in an inordinate number of unusual occurrences. Table 42 shows the extreme rosters and the number of unusual ranks occurring in each.

TABLE 42

<u>Roster Identification</u>	<u>Standard Deviation</u>	<u>Unusual Ranks</u>
<u>Uniform Rosters</u>		
Capacity: High - F5 [1;] (30)	2.85	2
Low - F3 [8;] (17)	1.28	0
Duration: High - F5 [7;] (36)	2.72	2
Low - F3 [8;] (17)	1.26	0
<u>Poisson Rosters</u>		
Capacity: High - P5 [3;] (62)	2.21	4
Low - P3 [4;] (43)	1.11	1
Duration: High - P5 [2;] (61)	2.09	1
Low - P3 [6;] (45)	1.04	3

From this, it is seen that the rosters with the most unusual standard deviations, either high or low, have about the expected number (2.7) of unusual occurrences. Therefore, there appears to be no relationship between standard deviation and frequency of unusual occurrences. It has also been shown at the start of this section that there is no relationship between job size and frequency of unusual occurrences.

3.7 Reliability of the General Rules

If a rule for identifying rosters which will behave unusually cannot be found, we are interested in the probability that the rule which has statistically been shown to be best will actually be best for any given roster and a specified measure of effectiveness. To find this probability, we find out how many times a roster produces an unusual result with the set of scheduling rules associated with

the primary rule which optimizes according to each measure of effectiveness. In other words, if the optimizing rule generally is LS, unusual occurrences with LS, LSSP, and LSLP are considered; and unusual results with the other twelve rules are ignored.

Table 43 shows the number of unusual occurrences for each roster with only the set of three optimizing scheduling rules considered for each measure of effectiveness. It in effect shows the number of times the best scheduling rule does not work.

TABLE 43

Number of Unusual Occurrences With
Optimizing Scheduling Rules

Roster	10	20	30	40	50	60
0						
1		2	4	1	1	
2				2		2*
3	3**	1*			1*	6
4	5		4			
5				1		
6						
7				1*		
8		2*			2	
9		1*		1*	2	

* Unusual according to measure A only.

** Unusual according to all measures.

Other - Unusual according to measures B, C, and D only.

It is interesting to note that with the exception of one roster, rosters which are not well behaved according to measure A are always well behaved according to measures B, C, and D; and rosters which are not well behaved according to B, C, or D are always so according to measure A.

This shows that the scheduling rules are fairly reliable. For measure A, only eight of sixty rosters show any extreme results; and these do not show extreme results according to the other measures. For criteria B, C, and D combined, only eleven of sixty rosters show any extreme results; and only four have more than three extremes which corresponds to one for each measure of effectiveness. A totally ill-behaved roster would have a total of twelve in its cell in Table 42, nine unstarred and three starred. Only two unstarred rosters exceed even half of nine, and two starred numbers exceed half of three.

Therefore, the scheduling rules are fairly reliable as they show a tendency not to work at all on only four out sixty or 7 percent of the rosters. The best scheduling rule has also been shown to be statistically significantly better than the others.

CHAPTER 4

PROGRAMMING OF SCHEDULING MODELS

4.1

4.1.1 The APL System

APL/360 is a version of Iverson's language executable on the IBM 360/67 computer at the University of Alberta.[1] The user has direct access to the computer via a terminal consisting of a modified typewriter, a small box of circuitry, and, in some cases, a dataphone. The computer can, therefore, be used at any place served by telephone lines.

To gain access to the system, the user turns the terminal on or if it is serviced by dataphone, dials the number of the telephone line allotted to the computer in the same way a standard telephone call is made. He then enters his identification number using the keyboard and is recognized by the system and given access to his private data and programs and the public libraries.

All programs and data are stored in "workspaces," and each user is assigned one or more of these. He can define a program and store it in his private workspace or can copy and store programs from the public libraries. The programs in the public libraries are also stored in similar workspaces. Each workspace is quite small, however, as it contains about 30,000 "bytes" while two or four bytes are required to store a number.

Each user has a library of workspaces which is identified by his identification number (the same one used to gain access to

the system). Each workspace within a user's private library and programs and data in the workspace are named by the user. The public libraries are similarly arranged with named workspaces and named programs within these workspaces.

Once a user is signed on by typing his number into the system, he frequently wants to do one of the things described below.

1. Use a previously defined workspace. If the workspace has previously been defined and labelled WSNAM, he types)LOAD WSNAM into the system. The system responds by indicating the date WSNAM was last used and stored and loads its contents into what is known as an "active workspace." There is one of these for each terminal. The contents of WSNAM are left unaltered as a result of following action on the terminal which only alters the contents of the active workspace.
2. Save the active workspace. The contents of the active workspace replace the contents of WSNAM as stored permanently in the computer when the command)SAVE WSNAM is entered into the system. The user's work in a current session essentially consists of alterations to the active workspace. If the "save" command above is not entered before the session is ended, the contents of the active workspace are erased when the terminal is turned off, and the work done in the session is lost.
3. Obtain another workspace. Apply to the computer centre for another workspace. At the next session type)SAVE ANOTHER which identifies the new workspace by the label ANOTHER. The system responds by typing ANOTHER SAVED "DATE" or if this is attempted

without requesting another workspace, by SAVE RATION EXCEEDED. The user then types)LOAD ANOTHER, and the contents of ANOTHER are loaded into the active workspace for use, modifications, and storage.

4. Copy a program or data from another workspace. If ANOTHER is currently loaded into the active workspace and a program PROGNAME is defined and stored in the other workspace WSPACE, the command)COPY WSPACE PROGNAME copies the program into the active workspace.

If a program to which access is required is stored in another user's private workspace and his number is NUMBER and his workspace name is WORKSPACE, the command)COPY NUMBER WORKSPACE PROGNAME transfers the program from the other user's private library to the active workspace.

If a public library is called LIBRARY and a workspace in this public library is called PACKAGE, the command)COPY LIBRARY PACKAGE PROGNAME is used to copy a program from the public library to the active workspace. Data is handled in the same manner as programs.

The procedure necessary to start up the system and prepare it for operation is, therefore, very simple. In spite of the small size of the individual workspace, it is possible to arrange each workspace so that only programs and data relevant to the problem at hand are actually stored in it.

Since a number of active workspaces are being processed at any one time, this is actually a time sharing system even though

each user has an illusion of continuous response. At present, there is space for 31 terminals to be processed by a no priority sequencing method.

4.1.2 Operating Techniques

Several features are incorporated in the system which make it possible to program, debug, and alter programs and data rapidly. Direct access to the computer makes it possible to try a program as soon as it is written and make alterations to it while it is being executed.

Once the system has been set up in a way described in 4.1.1, the user may wish to do the following operations:

1. Define a program. Type any ∇ PROGNAME which defines the program to be entered as PROGNAME. A[1] is typed by the system, and the user enters the first line of his program followed by a carriage return. A[2] is then typed and so on until the user enters the last line followed by ∇ .
2. Define data. If the item to be entered is to be labelled DATA, type DATA \leftarrow followed by the data to be so labelled.
3. Execute a program. Type the program name PROGNAME into the system. Provided all the input required by the program is defined, the program is executed and information is generated, stored, and displayed as required in the program. If an error is encountered, the execution is suspended, the offending statement is displayed with a diagnostic message, and control passed to the user so a correction may be made.
4. Display data. Type the data name DATA into the system.
5. Display a program. Type the command ∇ PROGNAME [0] ∇ into

the system. The program is displayed as it was entered but is not executed.

6. Correct a program. If the user wishes to correct statement N of program PROGNAME, he enters the command ∇ PROGNAME [N] followed by a carriage return. The system responds by typing [N] and the user then enters the corrected statement followed by a ∇ .

7. Resume execution of a program. If execution of a program has been suspended, a signal PROGNAME [N] is displayed at the time of suspension; and N is the statement number at which suspension occurred. To resume execution, type \rightarrow N followed by a carriage return.

8. Delete a program or data. Deletion is done using the command ∇ PROGNAME ∇ or ∇ DATA ∇ .

9. Stop execution of a program. A program can be suspended by pushing the ATTN button on the keyboard. This causes the statement PROGNAME [N] to be typed and control passes to the keyboard. The contents of the workspace can be stored, and the user can start the next session at the point he finished the last one by typing \rightarrow N. If the program is stopped and it is not desired to run it to completion, typing \rightarrow 0 until a Δ results in no output will clear the system.

The above indicates that the mechanics of the operation of the system permits very rapid programming due to instant error feedback and the ease with which corrections are made. A more detailed explanation for someone wanting to use the system is given in the reference manual.[1]

4.1.3 The APL Language

The APL language is so constructed that an algorithm can be expressed in a form close to its mathematical statement. A few symbols are predefined as operators while any other combination of letters can be used to identify variables or programs. In the execution of statements, a right to left precedence rule is employed except when overruled by parentheses. That is, the right-hand argument for any operator is the entire expression to the right, and the left-hand argument is the immediate symbol to the left.

Vectors and matrices are handled in the same way as numbers by most of the operators. Test loops for the end of the array are unnecessary greatly simplifying the programs compared to what would be required in some other programming languages. For example, the sum of two matrices is defined as another matrix consisting of the sums of corresponding elements. The APL command to add quantities A and B and store the result in C is $C \leftarrow A + B$. If A and B are matrices, this automatically specifies C according to this definition of matrix addition; and if A and B are different sizes, an error is indicated.

The operators in APL are either monadic or dyadic meaning one or two arguments, respectively, are required. The same symbol may mean two different things when used as a monadic or dyadic operator; for example, if A is a number and B is a vector, the operator \uparrow in the monadic expression $\uparrow A$ generates a vector and in the dyadic expression $B \uparrow A$ generates a number.

Basic to any programming language is the ability to

express mathematical relationships precisely and the ability to conditional branch. Each of these will be discussed in turn.

The basic dyadic symbols for the five elementary operations are $-$, $+$, \times , $-$, and $*$ where the latter means exponentiation. Parentheses () are used to enclose expressions in order to supercede the right to left rule. The dyadic symbol \leftarrow sets the left argument equal to the right argument.

The conditional branching is done using the standard symbols $< \leq = \geq$ and $>$ enclosed in the form $\rightarrow (A = B) / \text{STATEMENT}$. This means that if A equals B, the next statement to be executed is that labelled STATEMENT. Otherwise, the next statement in the program is executed whether it is labelled or unlabelled.

The ability to handle vectors and matrices is due to several other operators not found in some other languages. The number of elements in a vector A is ρA ; and if A has been previously defined, ρA is automatically set equal to the order of vector A. $A \dot{?} B$ is a vector of order A and all elements equal to B while $\dot{?} A$ is a vector of order A and elements 1, 2, ..., A. Certain elements are picked out of arrays very easily. For example, Γ/A and L/A are the maximum and minimum elements, respectively, of A. A certain element in a vector is located by the expression $A \dot{?} B$ which is the index of the first element of A equal to B. $A \dot{?} (\Gamma/A)$ is then the index of the maximum element of a vector.

The element or elements of a vector are called or identified by $A[]$ where the expression in the brackets is the index. It may be one number or an expression which produces integers. For example, $A[5;3]$ is the fifth and third element of

A, respectively. Also, $A[2 + 2B]$ represents the 3rd, 4th, 5th, ..., B, B + 1, B + 2 elements of vector A.

The variables in APL may be specified as either local or global at the time the program generating them is written. A global variable label exists for the entire workspace it is in, and execution of any program defining a variable using that label will redefine it. In contrast, if a variable is local within a program and the same label is used for a global variable in the same workspace, executing the program containing the local variable does not use or respecify the global variable in the workspace.

An example of local variables is illustrated by the program DIFFMEANS (Appendix A.3.a.). Variables after the semicolon are local, that is, they are defined and used within the function or program and not used outside of it.

Any program can be written as a monadic or dyadic operator with input and output arguments specified. The program GENFLT (Appendix A.1.a.) illustrates this point. The input arguments are MAX and NUM, and the command 20 GENFLT 50 is equivalent to specifying MAX = 20 and NUM = 50 and then executing the program. The output argument is V; and if the result is to be stored in RESULT, $RESULT \leftarrow 20 \text{ GENFLT } 50$ is equivalent to executing the program and then setting RESULT = V.

It is, therefore, very easy to specify the input and output arguments as this can be done in the same statement that orders the program to be executed. If a program requires more than two input arguments, the two labels which are available are specified as scalars, vectors, or matrices as required. An example

of this is DIFFMEANS where six inputs are required and are introduced via the two labels A and B. A and B are both vectors of order three containing three input arguments each.

4.1.4 Usefulness of APL

The APL system was used in the simulation model for three main reasons: ease of access, direct link with the computer, and its adaptability to the model.

1. Ease of access. Because the system is time shared, it is possible for many people to use the system at the same time. There was little trouble in getting adequate computer time.
2. Direct link to computer. This permitted the type of instant error feedback and instant correction feature described in Section 4.1.2. This was useful as the model required a large number of programs which are dependent on each other. If a day or two delay occurred between an error and its correction, it would have been impossible to program the model in the time allotted.
3. Adaptability to the model. A main feature of APL is its ability to handle arrays with a minimum of "housekeeping" programming to specify the arrays and flag the end of them. Section 4.2.2.c.ii illustrates this point in the explanation of the program SUMMARY.

As there was a large amount of data to be handled (the initial rosters contain 3,000 numbers) and similar operations are to be done on large groups of data, it is useful to store and use the data in matrix form. The matrix is then processed

in an orderly fashion, and this language is easily adapted to processing matrices.

Several disadvantages in the system, as it exists, were also found. Since the system is only in operation for a short time each day, it is desirable to be able to actually use the computer as much as possible while connected to the system; however, data must also be entered into the system, and this must be done while it is in operation. There is no provision for slow entry of data on a relatively inexpensive machine (such as a keypunch) which gets it into a form which can be rapidly entered into the main computer; however, the model was designed in such a way as to minimize the data handling requirements.

The second disadvantage lies in the small workspace size and this eliminated the Balas algorithm from further consideration. While the algorithm is not practical due to inherent difficulties (inefficient when matrix size exceeds 30×30), it could have been used to find optima which would be useful in checking the results from the heuristic rules. It may be possible to store 10,000 to 15,000 numbers in one workspace while the Balas algorithm requires a main matrix of over 100,000 numbers for the size of problem considered here. (See Section 2.2.2.)

In spite of these difficulties, the system was considered to be useful for this model, and the heuristic rules were simulated in APL and the results obtained from this system.

4.2 Programs Used in the Simulation

The programs used in this model fall into three categories:

1. Programs which generate and process the input data (rosters).
2. Programs which schedule the rosters and compute summary statistics from the schedules.
3. Programs which process the summaries generated above.

Each set of programs will be discussed in turn in Sections 4.2.1, 4.2.2, and 4.2.3.

4.2.1 Data Generators and Processors

The data generators are the routines GENPOISSON and GENFLT. The processors are CHECK, STRINGOUT, SUMMARY, and TABFREQ. These programs are displayed in Appendix A1.

GENPOISSON and GENFLT generate a vector of order L and mean M or a vector of order L and range 1 to R when the commands L GENPOISSON M or L GENFLT R are entered into the system with the workspace REGINA loaded into the active workspace. These programs are independent of any other privately defined or public library functions; therefore, they are executable without any other functions defined or loaded into the workspace.

The summary characteristics of the rosters are computed by typing SUMMARY followed by F3, 4, or 5; or P3, 4, or 5, depending on whether we wish to compute the characteristics of the uniform or the Poisson distributed rosters with mean 3, 4, or 5. For example, SUMMARYP5 computes the characteristics of the ten Poisson rosters with mean 5. The output is described in Section 4.3.3.

To use this routine, two conditions must be met:

1. The rosters must be in appropriate form
2. Several other programs must be defined

For point 1, the roster capacities must be in a matrix labelled ROSC and durations in a matrix labelled ROSD followed by the same suffix as SUMMARY. This matrix must have as many columns as elements in the roster (25) and as many rows as rosters (10). This is elaborated in Section 4.3.3.

For point 2, conditions defined by the hierarchy of programs stated in Table 44 must be met.

TABLE 44

Hierarchy of Programs in Workspace SASK

	CHECK	RND	STRINGOUT	SUMMARY	TABFREQ
CHECK		✓			✓
RND					
STRINGOUT					
SUMMARY	✓	✓			✓
TABFREQ					

A check (✓) indicates that the program on the vertical axis must be defined in order to execute the program on the horizontal axis. This convention will be kept in future hierarchy tables.

The programs STRINGOUT and TABFREQ are independent of any other programs, and no others depend on them as they have no (✓) in either vertical or horizontal columns. RND is independent of any others but others are dependent on it (checks in horizontal

column only). SUMMARY is dependent on other programs but none are dependent on it while CHECK is both dependent on other programs and others are dependent on it.

The rosters are checked using the program SUMMARY which feeds each roster in turn to the program CHECK by allocating each row of the roster matrices to the input arguments of CHECK. It accumulates the summary data generated by CHECK in a matrix SUMR followed by the same suffix as follows SUMMARY.

CHECK computes the ranges, averages, and standard deviations for the roster assigned to it and using RND rounds these off to a suitable number of places of decimal. It uses the program TABFREQ to compute how many of each element there are in the roster and prints the roster. It then passes control back to SUMMARY which stores the results in SUMR and assigns it another roster which is the next row of the matrices containing the rosters. The n'th row of SUMR contains the characteristics of the roster represented by the n'th row of the roster matrix. When all the rosters are processed, SUMMARY prints the summary characteristics and passes control back to the keyboard.

4.2.2 Main Scheduling Routine

4.2.2.a Main Routine

The scheduling of the roster and computation of the summary characteristics for each roster is coordinated by a program MAIN which accepts definition of the data (roster and shop), incorporates the scheduling routines, compiles data from them, and prints the summary statistics (STATS).

4.2.2.b Primary Scheduling Programs

The programs which incorporate the fourteen scheduling rules are called RANDOMS, RANDOMFITS, SSRANDOMFIT, LSRANDOMFIT, SPFITS, LPFITS, SPSCFITS, SPLCFITS, LPSCFITS, LPLCFITS, SSSPFIT, SSLPFIT, LSSPFIT, and LSLPFIT. These are the program names of the scheduling rules described in Section 1.5.1 and are in the same order as they are defined in Table 3. Table 45 shows the dependence of these routines on the sorter, scheduler, and summary routines which will be described in Section 4.2.2.c.

TABLE 45

Dependence of Primary Scheduling Programs

	ASGNXT	ASGNAHD	DECSORT	ASSORT	DECSORT 3	ASSORT 3	SUMARY
RANDOMS	✓						✓
RANDOMFITS		✓					✓
SSRANDOMFITS		✓				✓	✓
LSRANDOMFIT		✓			✓		✓
SPFITS		✓		✓			✓
LPFITS		✓	✓				✓
SPSCFITS		✓		✓			✓
SPLCFITS		✓	✓	✓			✓
LPSCFITS		✓	✓	✓			✓
LPLCFITS		✓	✓				✓
SSSPFIT		✓		✓		✓	✓
SSLPFIT		✓	✓			✓	✓
LSSPFIT		✓		✓	✓		✓
LSLPFIT		✓	✓		✓		✓

The seven routines upon which the fourteen scheduling programs are dependent fall into three categories and are described in Section 4.2.2.c.

1. Scheduler - ASGNXT, ASGNAHD
2. Sorter - DECSORT, ASSORT, DECSORT3, ASSORT3
3. Summary - SUMARY

The sequence of operations in a typical scheduling routine is to accept the input shop and roster defined by MAIN, rearrange the roster using 0, 1, or 2 sorters routines, schedule the jobs using one of the schedulers and compute summary statistics. As an example, the method is described in detail for one scheduling routine, LSSPFIT, which assigns jobs with priority to largest size and breaks ties according to smallest duration first.

FIGURE 10

LSSPFIT

Largest Size - Smallest Duration
Scheduling Program

- | | |
|---------------------------------|--------------------------|
| [1] CAP \leftarrow INPCAP | [7] DECSORT 3 |
| [2] DUR \leftarrow INPDUR | [8] X \leftarrow BB |
| [3] NRJOB \leftarrow INPNRJOB | [9] P \leftarrow AA |
| [4] ASSORT | [10] C \leftarrow SHOP |
| [5] DUR \leftarrow AA | [11] ASGNAHD |
| [6] CAP \leftarrow BB | [12] SUMARY |

Statements 1, 2, and 3 specify the roster from the INP variables defined by the program MAIN as global variables. In this way, each of the fourteen scheduling routines including this one is guaranteed an unaltered set of input data.

Statement 4 sorts the roster in ascending duration order as required by LSSP. Note that this is the tie-breaking sort and is done first. Statements 5 and 6 place the roster capacity and duration vectors which result from [4] in suitable order for statement 7.

Statement 7 sorts the roster in descending size order which is the primary sort required by LSSP. Because the sorting method chooses the first number in the sequence in case of ties, the second sort has already been done according to the specified tie-breaking rule (in statement 4). Statements 8 and 9 put the roster, which is now in order according to the required priority, into the input arguments of the scheduler routine; and [10] specifies the shop the same as that specified by MAIN.

Statement 11 is the scheduler routine which incorporates the fitting procedure and produces the schedule. Statement 12 computes summary statistics from the schedule produced.

4.2.2.c Dependent Sorter, Scheduler, and Summary Routines

As indicated by Table 45 in 4.2.2.b, these routines are necessary for the execution of the main scheduling routines. Each of these will be described in turn.

4.2.2.c.i Sorter Routines

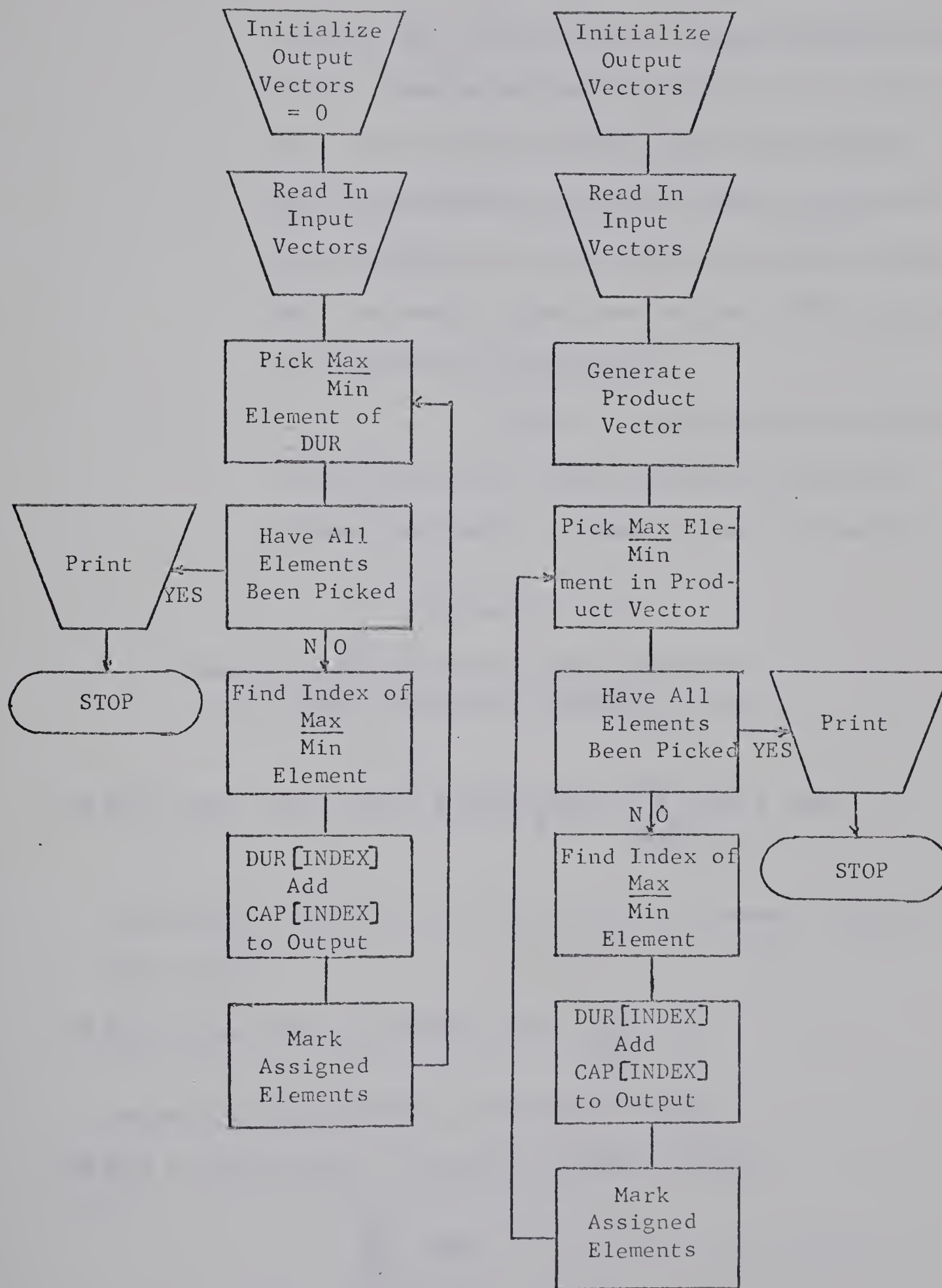
These consist of four programs -- DECSORT, ASSORT, DECSORT3, and ASSORT3, none of which are dependent on other predefined functions. The input arguments are vectors DUR and CAP, and the output is vectors AA and BB. These contain the same elements as inputs DUR and CAP but are rearranged in some specific order. If $DUR [I]$ becomes $AA [J]$, $CAP [I]$ will become $BB [J]$ for all I and J.

The routines ASSORT and DECSORT rearrange the input vector DUR in ascending or descending order, respectively, while ASSORT3 and DECSORT3 generate a new vector SIZE such that $SIZE [I] = DUR [I] \times CAP [I]$ for all I and then sort the vector SIZE in ascending or descending order, respectively.

Again, if $DUR [I] \times CAP [I]$ generate $SIZE [I]$ and $SIZE [I]$ becomes $SIZE [J]$ after sorting, then $DUR [I]$ and $CAP [I]$ become $AA [J]$ and $BB [J]$ for all I and J. The sequence is shown in Figure 10.

FIGURE 10

Flowchart for Assort/Decsort



4.2.2.c.ii Summary Routine

This is one program SUMMARY which computes the nine measures of effectiveness according to the formulae outlined in Section 3.2.1, puts them in a vector called SUM, and prints this vector. (In addition, MAIN adds the vector SUM calculated for each schedule to a matrix which is further worked into the ranked summary statistics.) This program is independent of any others.

Figure 11 compares the APL statements in Section 3.2.1 to more conventional notation showing the limits of summation over the arrays.

Figure 11

Comparison of APL Summary Calculations to
More Conventional Notation

$$\text{SUM [1]} = \text{mean start time} = (+/\text{NUM}) \div \rho \text{ NUM} = \frac{\sum_{i=1}^J \text{NUM } i}{J} = \text{MS}$$

where NUM i is start time of i 'th job J is number of jobs in the roster.

$$\text{SUM [2]} = \text{time shop is occupied} = \uparrow/F = \text{Max. } F_i \quad 1 \leq i \leq J$$

where F_i is the finish time of the i 'th job.

$$\text{SUM [3]} = \text{average jobs in process} = (+/\text{JOB}) \div (\uparrow/F) =$$

$$\frac{\sum_{i=1}^N \text{JOB}_i}{\text{Max. } F_i}$$

where JOB_i is the number of jobs running in time i

N is the number of time periods any job is running

SUM [4] = average efficiency = $+(P \times X) \div (CC \times \lceil F \rceil)$

$$= \frac{\sum_{i=1}^J P_i X_i}{CC \cdot (\text{Max } F_i)} \quad 1 \leq i \leq J$$

where P_i and X_i are the durations and capacities of the i 'th job

CC is the initial shop capacity in each period

SUM [5] = mid-range efficiency = $1 - \text{SLACK} \div CC \times 2 \times \text{BEGIN}$

$$= 1 - \frac{\text{SLACK}}{2 \cdot (CC) \cdot (\text{BEGIN})}$$

where $\text{BEGIN} = ((\lceil F \rceil - (4 \mid F)) \div 4$

= next lowest max. F_i that is divisible by 4

divided by 4

$\text{SLACK} = +/C [\text{BEGIN} + 2 \times \text{BEGIN}]$

$$= \sum_{i = \text{BEGIN} + 1}^{2 \cdot \text{BEGIN}} C_i \quad , \quad C_i = \text{unused shop capacity in period } i$$

The notation $A \mid B$ represents the remainder from the operation

$B \div A$; e.g., $4 \mid 9 = 1$

SUM [6] = weighted delay factor = $+/P_X X_X (\text{NUM} - 1) = \sum_{i=1}^J P_i X_i d_i$

where d_i is the delay of job i

SUM [7] = total start time = $+/ \text{NUM} = \sum_{i=1}^J \text{NUM}_i$

SUM [8] = start time standard deviation =

$$\begin{aligned} & ((+/(\text{NUM} - (\rho \text{NUM}) \rho \text{MS})) * 2) \div \rho \text{NUM}) * .5 \\ & = \sqrt{\frac{\sum_{i=1}^J (\text{NUM}_i - \text{MS})^2}{J}} \end{aligned}$$

where MS is mean start time defined in SUM [1]

The expression $(\rho_{\text{NUM}}) \rho_{\text{MS}}$ develops a vector of length

$\text{NUM} = J$ and each element MS . The subtraction $\text{NUM} - \text{MS}$

would not be possible directly as these vectors must have

equal lengths for an element by element subtraction to be made.

$\text{SUM}[9] = \text{start time deviation about } 0 = ((+/\text{NUM} * 2) \div \rho_{\text{NUM}}) * 0.5$

$$= \sqrt{\frac{\sum_{i=1}^J \text{NUM}_i^2}{J}}$$

4.2.2.c.iii Scheduling Routines

These are the key routines in the model and are dependent on three routines which provide more detail about the generated schedule which will be described in Section 4.2.2.d.

ASGNXT and ASGNAHD require as inputs a vector representing shop capacity (C) and vectors representing the durations (P) and capacities (X) of the jobs in the roster. The routines will assign each job i (P_i , X_i) to the shop so that the sum of the capacity requirements of all jobs running does not exceed the shop capacity at any time. The output is a schedule; i.e., a vector of start times (NUM) and finish times (F) for the roster, a vector (C) indicating unused shop capacity in each time period and a vector JOB indicating the number of jobs in process in each time period.

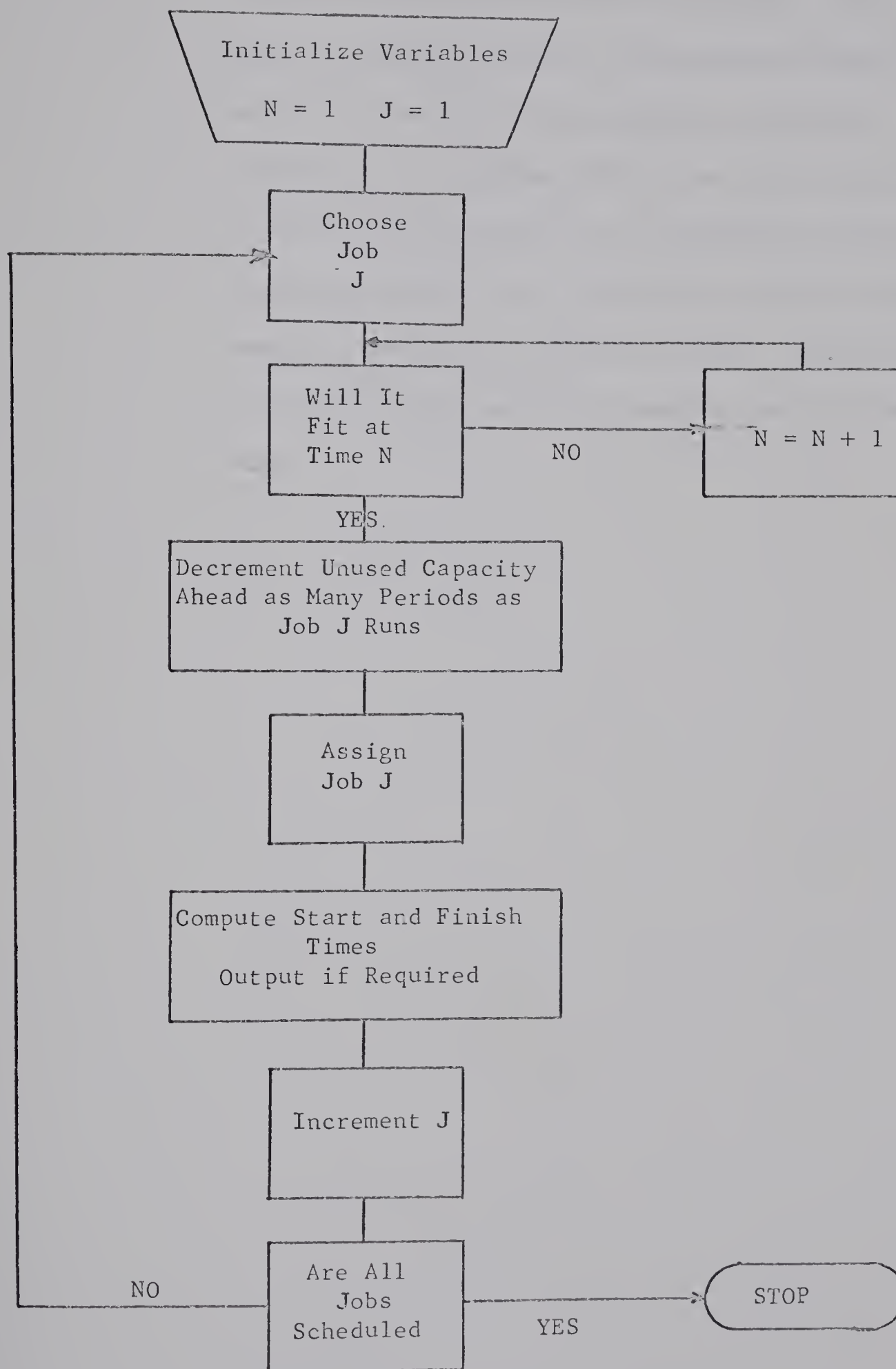
The routine ASGNXT schedules the jobs in the same order as they appear in the input roster.

At the first time period $t = 1$, the first A jobs such that:

$$\sum_{i=1}^A X_i \leq C_1 \text{ and } \sum_{i=1}^{A+1} X_i > C_1$$

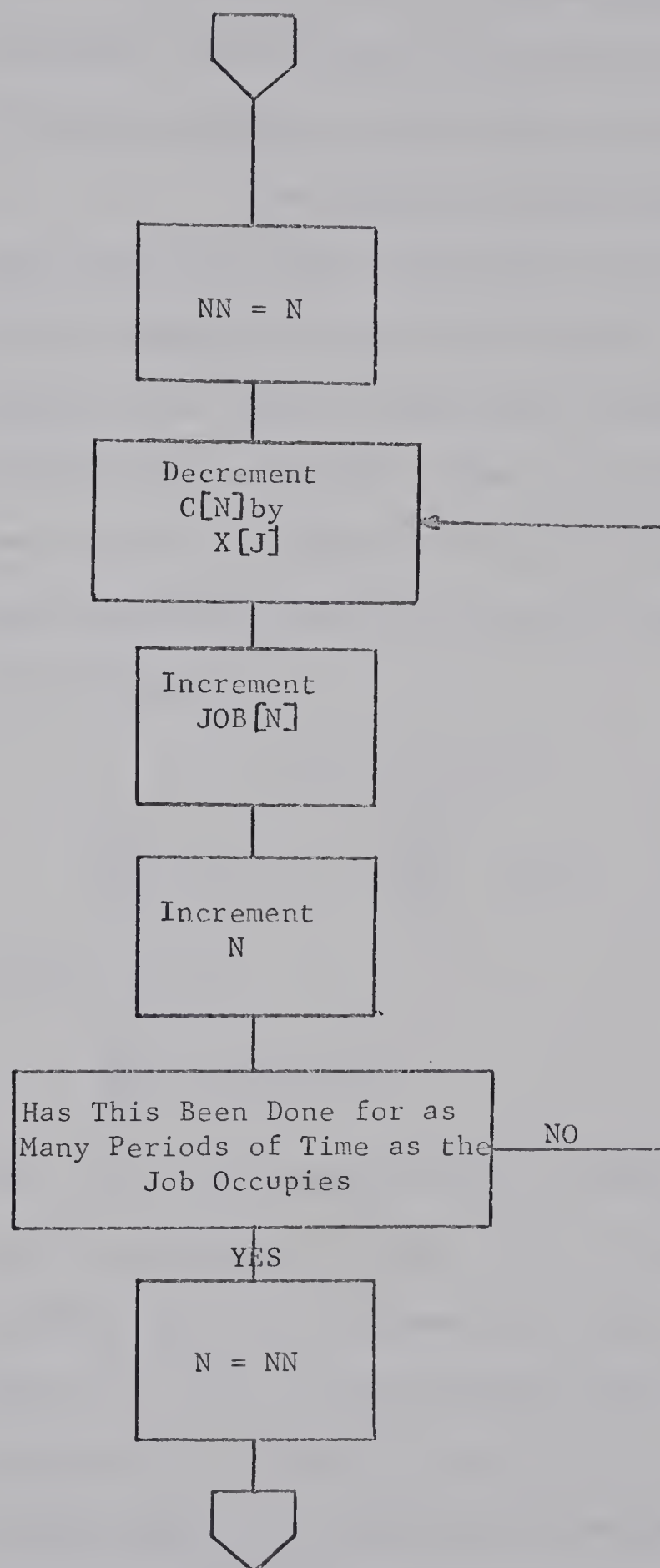
are scheduled. Then the clock is advanced to $t = 2$ and the $A + 1$ 'th job is tried. If there is less unoccupied capacity at $t = 2$ than the requirement of the $A + 1$ 'th job, the clock is again advanced; and the $A + 1$ 'th job is tried again. This continues until it is scheduled. This procedure is repeated with each succeeding job in the roster in order until they are all scheduled. Figure 12 illustrates the procedure diagrammatically.

FIGURE 12

Flowchart for ASGNXT

The variable NN is introduced to mark the time period in which each assignment occurs. When the unused capacity is decremented beyond the period of time in which the job is initially assigned, N is incremented; but we wish to go back to the time period the job is assigned to see if the next job will fit. Otherwise, only one job would be assigned in any time period. Figure 13 illustrates this method of keeping track of the base time.

FIGURE 13

Decrementing Procedure for N

The output format from ASGNAHD is the same as ASGNXT, but the scheduling method is different. This routine is the basis of the fitting procedure mentioned several times previously.

An attempt is still made to schedule the jobs in the order they appear in the input roster; however, if a job will not fit, a search ahead in the roster is made until the shop is filled or the roster exhausted. Then the clock is advanced to the next time period, and the first job which was encountered last time period but was not scheduled is tried again.

If the situation

$$\sum_{i=1}^A X_i < C_1 \quad \text{and} \quad \sum_{i=1}^{A+1} X_i > C_1$$

occurs, we try

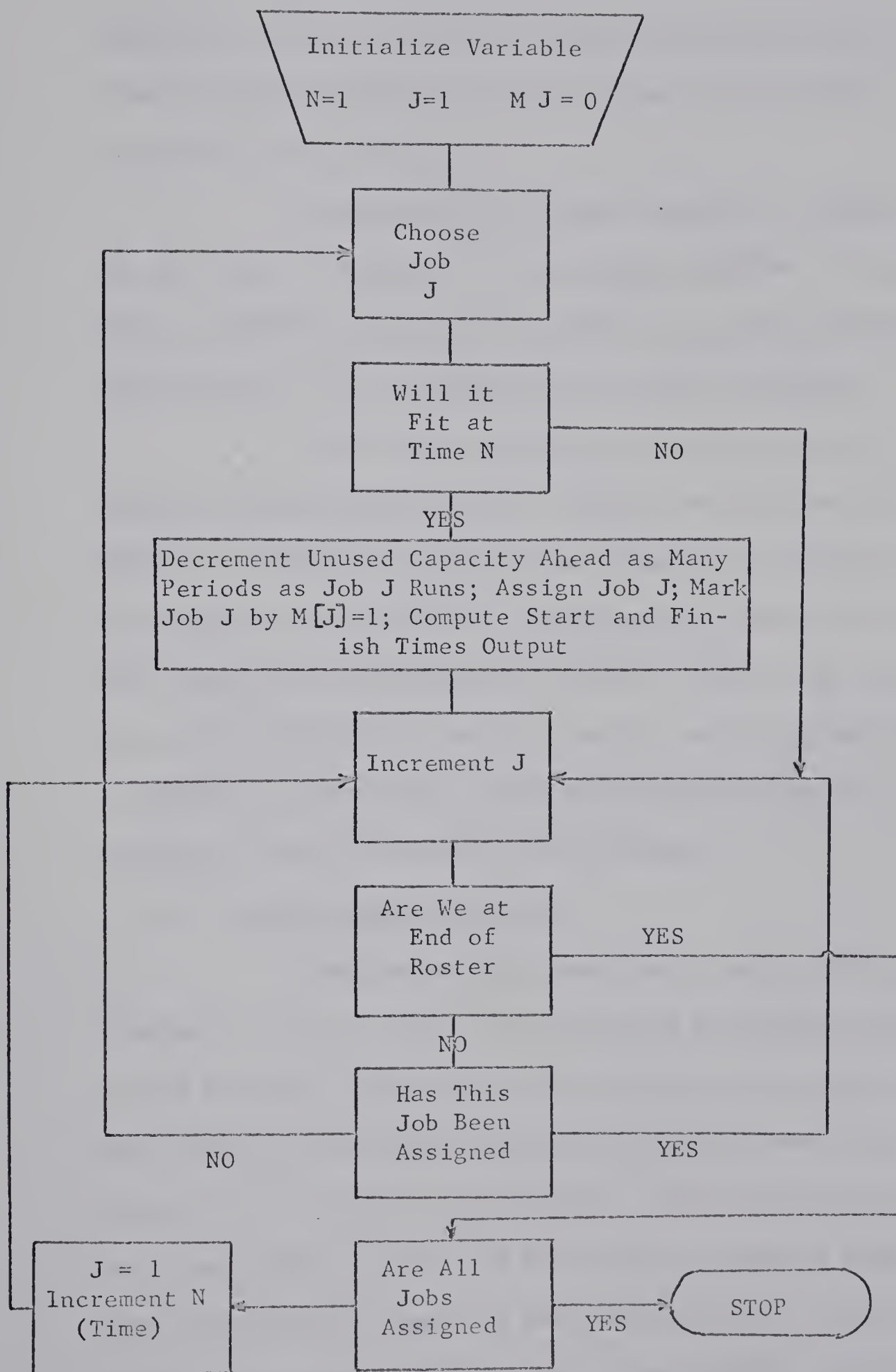
$$\sum_{i=1}^A X_i + X_{A+2} \text{ ? } C$$

If ? is $<$, we schedule job $A + 2$ and consider X_{A+3} in the same manner as above. If ? is equal, we schedule job $A + 2$, increment the clock, and try with job $A + 1$. If ? is $>$ we ignore job $A + 2$ and try job $A + 3$. Jobs $A + 2$, $A + 3$, ..., J are tried in this manner; and when job J is reached, the clock is incremented, and the first unassigned job in the roster tried again.

To avoid having jobs assigned more

than once, each job is initially marked with a 0; and when it is scheduled, the mark is changed to a 1. Before the procedure above is tried, the appropriate element of the marking vector is tested for 0 or 1. If it is 0, an attempt is made to schedule the job; and if it is 1, the job has already been scheduled and the next job in the roster is considered. Figure 14 illustrates the ASGNAHD algorithm.

FIGURE 14

Flowchart for ASGNAHD

4.2.2.d Roster Detail Routines

These routines can be added to ASGNXT and ASGNAHD to provide more detail about the schedule than its summary characteristics produced by the program SUMARY described in 4.2.2.c.ii.

Statement [1] of either ASGNAHD or ASGNXT can be either a $\rightarrow 2$ or call for program HEADING. Statement [16] of ASGNXT can be a $\rightarrow 17$ or call for program DISPLAY and statement [17] of ASGNAHD can be $\rightarrow 18$ or DISPLAY.

If the \rightarrow alternative is chosen, only the summary characteristics of the schedule are printed; if HEADING and DISPLAY are used, the schedule is printed in the order the jobs are scheduled including the start and finish time, capacity, and duration of all the jobs in the roster. The routine ASGNDETL is attached at the end of either ASGNAHD or ASGNXT and causes the unused capacity and number of jobs running in each time period to be printed.

4.2.2.e Summary Ranking Routine

The program MAIN generates a matrix MAT consisting of all the summary data produced by SUMARY after each of the fourteen scheduling routines is applied to the roster. The ranking routine STATS applies the ranking method of Section 3.4.1 to this 9 x 14 matrix. STATS scans each of the nine columns in turn and picks out the maximum (MAX) and minimum (MIN) element in each and finds their difference (DIFF). For each element in the column, $\frac{\text{ELEMENT} - \text{MIN}}{\text{DIFF}}$ is

TABLE 46 (Cont'd.)

TABLE 40 (cont'd.)																											
	MAIN	RANDOMS	RANDOMFITS	SSRANDOMFIT	LSRANDOMFIT	SPFITS	LPFITS	SPSCFITS	SPLCFITS	LPSCFITS	LPLCFITS	SSSPFIT	SSLPFIT	LSSPFIT	LSLPFIT	ASGNXT	ASGNAHD	DECSORT	ASSORT	DECSORT3	ASSORT3	SUMARY	STATS	HEADING	DISPLAY	ASGNDETL	RND
ASSORT																											
DECSORT3																											
ASSORT3																											
SUMARY																											
STATS																											
HEADING																											
DISPLAY																											
ASGNDETL																											
RND																											

(✓) compulsory dependence

0 optional dependence (see Section 4.2.2.d)

4.2.3 Sumary Processing Programs

The key program in this group of programs is labelled SYN prefixed by A, B, C, or D, corresponding to the four measures of effectiveness. This is the program which generates the chart of means and standard deviations which is displayed in Table 14 of Section 3.4.3 and is the main result of this experiment.

The program SYN accpets the rankings computed by STATS for each scheduling rule for each measure of effectiveness in matrix form, essentially a 60 x 14 matrix for each measure of effectiveness. This corresponds to sixty rosters and fourteen scheduling rules run together. The output is a matrix MSD()TOTAL where () contains the same letter A, B, C, or D as prefixed SYN corresponding to the measure of effectiveness. For example, BSYN computes the mean and standard deviation of the shop occupied time (measure B) for the sixty rosters for each of the fourteen scheduling rules and places the number of items (60), mean and standard deviation, in a 3 x 14 matrix MSDETOTAL.

SYN is dependent on a program MSD which performs the actual calculations required. The statement MSD VECTOR yields a vector consisting of the number of elements in VECTOR, their mean, and their standard deviation calculated according to

$$\sqrt{\frac{\sum_{i=1}^N (\text{VECTOR}_i - \overline{\text{VECTOR}})^2}{N}}$$

where N is the number of elements in VECTOR

$\overline{\text{VECTOR}}$ is the mean of the elements in VECTOR

The chart of t values for the differences between means in MSD()TOTAL is generated by a program labelled COMPARE. This compares the first row of MSD()TOTAL to the other rows in turn, calculating t values for the differences between means then compares the second row to all subsequent rows and so on until all possible pairs of means are compared.

This program is dependent on a routine DIFFMEANS and is really an orderly method of feeding the input arguments to DIFFMEANS. This program incorporates a routine for calculating a t value from N_1 , S_1 , X_1 , N_2 , S_2 , and X_2 where N is the number of items in the sample; S is the standard deviation, and the X's are the means for which we are trying to establish significance of difference.

The formulae are:

$$Z = N_1 + N_2 - 2 \quad [7]$$

$$S = \sqrt{\frac{N_1 S_1^2 + N_2 S_2^2}{Z}} \quad [8] \quad 7-5.5$$

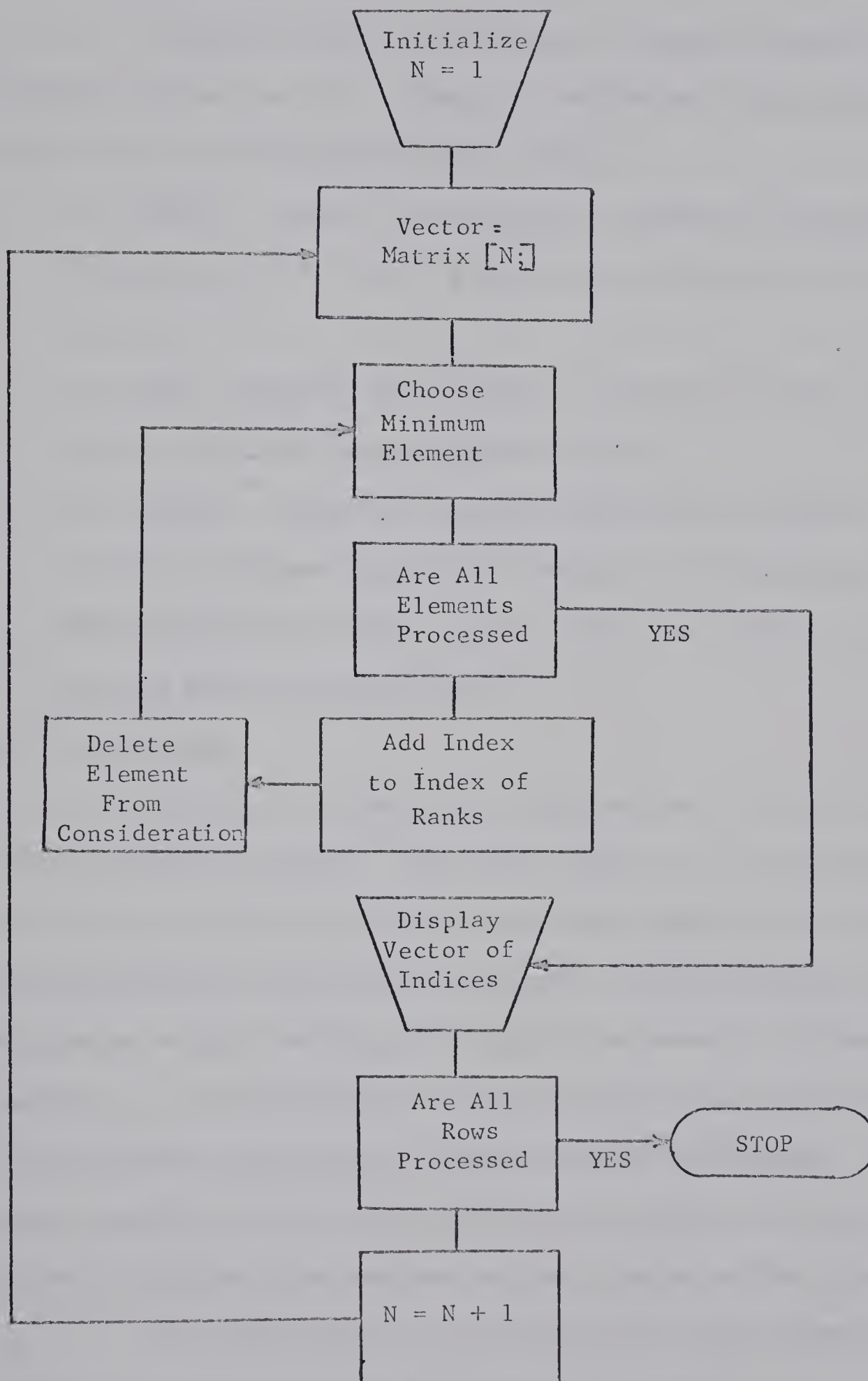
$$S_D = S \sqrt{\frac{N_1 + N_2}{N_1 N_2}} \quad [9] \quad 7-5.8$$

$$T = \frac{X_1 - X_2}{S_D} \quad [10] \quad 7-5.1$$

where the numbers in [] indicate the statement number in DIFFMEANS (see Appendix A.3.a) and following is the formula number in the reference.[7]

The program RANK is independent of other programs and there is no routine dependent on it. Given an initial vector, this will arrange the elements in ascending order and print only the indices which the elements had in the initial vector in this order. The input is a row of a matrix, and each row is automatically processed in turn. The procedure is illustrated in Figure 15. VECTOR is the vector to be ranked.

FIGURE 15

Flowchart for RANK

4.3 Data Handling

4.3.1 Workspace Allocation

The final model is contained in three workspaces containing programs and data. These are defined as follows where the block letters indicate the workspace name.

1. REGINA - contains the scheduling routines described in Section 4.2.2, the roster generating routines and rosters in process.
2. SASK - contains the routines for checking rosters, the rosters and their summary characteristics.
3. YORKTON - contains the t-test and ranking routines. The resulting rankings for the four measures of effectiveness are also stored here as well as their means and standard deviations for the group of sixty rosters.

4.3.2 Data Flow

The data is generated in the workspace REGINA from the roster generator programs. From this, there is a two-way branch; the rosters are stored in another workspace (SASK) due to lack of storage capacity in the initial workspace, and the summary results are dumped on paper while each roster is processed. The summary rankings for the four measures of effectiveness chosen are then re-entered into the system in another workspace (YORKTON). This could possibly have been done automatically while each roster was processed; however, the need was not anticipated at that time.

The data flow in each workspace was well automated. In SASK, the rosters were passed automatically through their checking routine and the results printed and stored for future access. In

YORKTON, the mean and standard deviation were computed from the summary data and printed and stored in matrices. The result of the t-test was dumped on paper only; however, to retrieve this, the user has only to execute the program ()SYN where () contains either A, B, C, or D, depending on the measure of effectiveness desired.

4.3.3 Data Storage

Advantage was taken of the ability of APL to handle matrices by storing and handling data in matrix form. An exception to this is the initial processing of each roster through the scheduling routines. In this case, each roster was processed individually since the operation frequently took ten to twenty minutes per roster. As the allowable session on a terminal is twenty minutes, it is pointless to have a system which automatically proceeds to the next roster. However, a minor modification to the input part of MAIN would enable this procedure to be automated.

In general, the model is arranged so that any procedure will take from five to ten minutes and then give control back to the operator. If the time was any shorter, a high proportion of the terminal session would be spent on "housekeeping" and if it were longer, there would be a lack of error control and the frequent nuisance of suspending programs when the session is over resulting in a chopped up output.

When the rosters were transferred to SASK, they were put into matrices which were identified by the letters ROS. The capacities and durations were put in separate matrices, and each set of two matrices contained the ten rosters which were generated

by the same process. Table 47 shows the names of the vectors in which the rosters are stored.

TABLE 47
Roster Storage Vector Names

<u>Roster Class</u>		<u>Capacities</u>	<u>Durations</u>
Uniform distributed	1 to 5	ROSCF3	ROSDF3
Uniform distributed	1 to 7	ROSCF4	ROSDF4
Uniform distribution	1 to 9	ROSCF5	ROSDF5
Poisson distribution*	Mean 3	ROSCP3	ROSDP3
Poisson distribution*	Mean 4	ROSCP4	ROSDP4
Poisson distribution*	Mean 5	ROSCP5	ROSDP5

The coding in Table 47 is interpreted as follows:

Indicates a roster 
 C - capacity, D - duration
 F - uniform, P - poisson

* See Section 3.1

All roster matrices are, therefore, of size 10 rows (representing ten rosters) and 25 columns (representing 25 jobs per roster).

The command SUMMARY Y(YZ) automatically computes the summary characteristics of the group of ten rosters $\overline{ROSC(YZ)}$ and places the summary in a matrix SUMR(YZ). The capacity average, standard deviation, maximum, minimum; duration average, standard deviation, maximum, minimum; and roster work required are stored in the rows of these matrices and, therefore, the matrix size is ten rows (representing ten rosters) by nine columns (representing the nine items above).

The summary data for each roster generated by the main processing routine was entered in YORKTON in matrix form. Each matrix represents ten rosters in the same groupings as before and one measure of effectiveness. Each matrix contains fourteen rows representing the fourteen scheduling rules and ten columns representing ten rosters in the group. The data is stored in matrices labelled as in Table 48.

TABLE 48

Summary Data Matrix Storage Labels

<u>Roster Type</u>		<u>Measure of Effectiveness</u>			
		A	B	C	D
Uniform	1 to 5	AF3	BF3	CF3	DF3
Uniform	1 to 7	AF4	BF4	CF4	DF4
Uniform	1 to 9	AF5	BF5	CF5	DF5
Poisson*	Mean 3	AP3	BP3	CP3	DP3
Poisson*	Mean 4	AP4	BP4	CP4	DP4
Poisson*	Mean 5	AP5	BP5	CP5	DP5

* See Section 3.1

The ranking matrix for any roster is contained in these summary matrices. For example, if we wanted the ranking matrix for the seventh Poisson distributed roster of mean 4, we would take the seventh column of AP4, BP4, CP4, and DP4.

The average rank of each scheduling rule according to each measure of effectiveness is stored in the second column of the matrices MSD(W)TOTAL where W = A, B, C, or D, depending on the measure of effectiveness. The first column of this matrix indicates the

number of items in the sample; and the third is their standard deviation. These are the matrices displayed in Table 14.

A summary of the data in storage is then:

Location	Data	Matrix Name
SASK	Rosters	ROS (X YZ)
SASK	Roster Summary	SUM R(YZ)
YORKTON	Rankings	(WYZ)
YORKTON	Ranking Summary	MSD(W)TOTAL

where the variables denote the following as mentioned above

W - measure of effectiveness

X - capacity or duration

Y - poisson or uniform

Z - mean of the distribution

CHAPTER 5

FURTHER RESEARCH

In this paper, only a very special case of a wide variety of scheduling problems was studied; and practical rules of thumb were developed for dealing with this case.

However, the programs are sufficiently general that with some modifications, other studies could be undertaken. The data storage system is such that comparisons of performance of the same rule operating on rosters generated in different ways could be explored. This was initially attempted, but the sample sizes were too small to get significant results. It may be that some rules will show greater or lesser effectiveness depending on whether the rosters are Poisson or uniformly distributed or are generated by some other process.

The field of dynamic scheduling problems (see Section 1.3) was not explored. The current model could be made to simulate a continual flow of jobs into the shop by stopping the process at suitable time periods, injecting a number of jobs generated in a specified manner, resorting the roster and starting over. The major trouble Conway found with the shortest operation discipline would now occur; i.e., large jobs would have very long waiting times. It would not have the very strong superiority in optimizing mean start time which was observed here.

The capacity of the shop could change over time, and it could also run for only a fixed number of time periods. An increasing shop could be handled without modification while for a decreasing shop, the main scheduling model would have to be modified to stop a job of greater

capacity requirements than the new shop capacity being scheduled too close to the change. The model is not set up to deal with a shop of fixed restrictive duration. An integer programming formulation may be required for problems involving shops of decreasing capacity or restricted duration.

A situation could occur where it is possible to "crash" a job at a certain expense. The expense could be such that if say a capacity = 5, duration = 5, job is crashed, it may require capacity = 7, duration = 4; the expense would be $28 - 25 = 3$ capacity time units. As with the restrictive shop, the model is not equipped to handle this. This would probably be an interesting study as "crashing" is frequent in practice; and some simple rules given the penalties for crashing may be developed based on simple conditions existing at the time a decision is to be made. Profitability of expanding or shutting down part of the shop could also be explored.

The work done in this paper has shown that simple rules can be applied to a job shop scheduling problem where the following conditions apply:

- a. The shop is of fixed capacity and nonrestrictive in duration.
- b. The entire roster is fixed and known at the beginning of the scheduling operation.
- c. The jobs are of unalterable capacity and duration.

These conditions are probably too strict for the model to have any direct practical importance. However, they should indicate the direction for further research to follow by giving future researchers a "feel" for what may be the best rule when the restrictions are lifted.

BIBLIOGRAPHY

- 1 Adams, W. S., An Introduction to APL/360, Department of Computing Science, University of Alberta, Publication No. 7, September, 1967.
- 2 Balas, Egon, "An Additive Algorithm for Solving Linear Programs With Zero-One Variables," Operations Research, Vol. 13, pp. 516-549.
- 3 Brooks, G. H., and White, C. R., "An Algorithm for Finding Optimal or Near-Optimal Solutions to the Production Scheduling Problem," Journal of Industrial Engineering, Vol. 16, pp. 34-40.
- 4 Conway, R. W., "Priority Dispatching and Work in Process Inventory in a Job Shop," Journal of Industrial Engineering, Vol. 16, pp. 123-130.
- 5 Fabrycky, W. J., and Shamblyn, J. E., "A Probability Based Sequency Algorithm," Journal of Industrial Engineering, Vol. 17, pp. 308-312.
- 6 Freund, J. E., and Williams, F. J., Modern Business Statistics, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1958.
- 7 Richmond, S. B., Statistical Analysis, Second Edition, The Ronald Press Company, New York, 1964.

APPENDIX A

A-1 Data Generators and Processors

A-1a. Generators

```

      ∇ GENFLT[ ] ∇
    ∇ V ← MAX GENFLT NUM; K; I
  [1] V ← 0; 0
  [2] I ← 0
  [3] GENFLT1: K ← ? MAX
  [4] V ← V, K
  [5] I ← I + 1
  [6] → (I < NUM) / GENFLT1
    ∇

```

```

      ∇ GENPOISSON[ ] ∇
    ∇ V ← LAMBDA GENPOISSON N; I
  [1] V ← 0; 0
  [2] I ← 0
  [3] GENPOISSON1: → (N < I + 1) / GENPOISSON4
  [4] U ← ( ? 100000 ) ÷ 100001
  [5] A ← B ← 1
  [6] K ← 0
  [7] F ← * ( - LAMBDA )
  [8] GENPOISSON3: → ( ( F × A ) > U ) / GENPOISSON2
  [9] A ← A + B ← B × LAMBDA ÷ K ← K + 1
  [10] → GENPOISSON3
  [11] GENPOISSON2: V ← V, K
  [12] → GENPOISSON1
  [13] GENPOISSON4: → 0
    ∇

```

Documentation

- * A vector V is generated. The elements of V have either a uniform or Poisson distribution
- * GENFLT specifies a uniform distribution with range of one to MAX. The order of the vector is NUM
- * GENPOISSON specifies a Poisson distribution with mean LAMBDA. The order of the vector is N

A - 1. b Processors

```

      ∇ SUMMARYF3[ ] ∇
∇ SUMMARYF3
[1]  N←1
[2]  SUMRF3←(10 9)ρ 0.
[3]  SUMMARYF31:N
[4]  SUMRF3[N;]←ROSCF3[N;]CHECK ROSDF3[N;]
[5]  N←N+1
[6]  →(N=11)/0
[7]  →SUMMARYF31
∇

```

```

      ∇ SUMMARYF4[ ] ∇
∇ SUMMARYF4
[1]  N←1
[2]  SUMRF4←(10 9)ρ 0
[3]  SUMMARYF41:N
[4]  SUMRF4[N;]←ROSCF4[N;]CHECK ROSDF4[N;]
[5]  N←N+1
[6]  →(N=11)/0
[7]  →SUMMARYF41
∇

```

```

      ∇ SUMMARYF5[ ] ∇
∇ SUMMARYF5
[1]  N←1
[2]  SUMRF5←(10 9)ρ 0
[3]  SUMMARYF51:N
[4]  SUMRF5[N;]←ROSCF5[N;]CHECK ROSDF5[N;]
[5]  N←N+1
[6]  →(N=11)/0
[7]  →SUMMARYF51
∇

```

Documentation

- * Computes summary statistics for the uniform rosters
- * Requires input matrices ROS CF and ROS DF representing capacities and durations of the jobs in the rosters
- * Generates an output matrix SUM RF consisting of the summary characteristics of the input rosters


```

      ∇ SUMMARYP3[ ] ∇
∇ SUMMARYP3
[1]  N←1
[2]  SUMRP3←(10 9)ρ0
[3]  SUMMARYP31:SUMRP3[N;]←ROSCP3[N;]CHECK ROSDP3[N;]
[4]  N←N+1
[5]  →(N=11)/0
[6]  →SUMMARYP31
∇

```

```

      ∇ SUMMARYP4[ ] ∇
∇ SUMMARYP4
[1]  N←1
[2]  SUMRP4←(10 9)ρ0
[3]  N
[4]  SUMMARYP41:SUMRP4[N;]←ROSCP4[N;]CHECK ROSDP4[N;]
[5]  N←N+1
[6]  →(N=11)/0
[7]  →SUMMARYP41
∇

```

```

      ∇ SUMMARYP5[ ] ∇
∇ SUMMARYP5
[1]  N←1
[2]  SUMRP5←(10 9)ρ0
[3]  SUMMARYP51:N
[4]  SUMRP5[N;]←ROSCP5[N;]CHECK ROSDP5[N;]
[5]  N←N+1
[6]  →(N=11)/0
[7]  →SUMMARYP51
∇

```

Documentation

- * Computes summary statistics for the Poisson rosters
- * Requires input matrices ROS CP and ROS DP representing capacities and durations of the jobs in the rosters
- * Generates an output matrix SUM RP consisting of the summary characteristics of the input rosters


```

      VCHECK[ ]V
    V A←X CHECK P
[1]  A←9ρ0
[2]  A[1]←(+ /X)÷ρX
[3]  VECTAVCAP←(ρX)ρ(A[1])
[4]  A[2]←4 RND(((+/(X-VECTAVCAP)*2)÷ρX)*0.5)
[5]  A[3]←L /X
[6]  A[4]←I /X
[7]  A[5]←(+ /P)÷ρP
[8]  VECTAVPRO←(ρP)ρ(A[5])
[9]  A[6]←4 RND(((+/(P-VECTAVPRO)*2)÷ρP)*0.5)
[10] A[7]←L /P
[11] A[8]←I /P
[12] A[9]←+/(P×X)
[13] ( ' ' )
[14] ( 'CAPACITIES ARE ' ; X ; )
[15] 'FREQUENCIES'
[16] TABFREQ X
[17] ( ' ' )
[18] ( 'DURATIONS ARE ' ; P ; )
[19] 'FREQUENCIES'
[20] TABFREQ P
[21] ( ' ' )
[22] A
[23] ( ' ' )
[24] ( ' ' )
[25] ( ' ' )

```

Documentation

- * Incorporated in the SUMMARY programs to perform the actual computation of the summary characteristics
- * Input is two vectors, X and P. X is the vector of capacities of the jobs in the roster, and P is the duration vector
- * Output is a vector A consisting of the nine summary results
- * Displays the vector A of summary characteristics
- * Incorporates the routine TABFREQ to display a frequency distribution of the capacities and durations of the jobs in each roster


```

      ∇ TABFREQ[ ] ∇
∇ TABFREQ R;S;I
[1]  I←1
[2]  S←100ρ0
[3]  TABFREQ1:S[I]←+/R=I
[4]  I,S[I]
[5]  →((+/S)=ρR)/0
[6]  I←I+1
[7]  →TABFREQ1
      ∇

```

Documentation

- * Input is vector R
- * Computes vector S, where S [I] is the number of elements of R equal to I
- * Displays I and S [I]

```

      ∇ STRINGOUT[ ] ∇
∇ STRINGOUT ROS
[1]  VECT←0ρ0
[2]  N←1
[3]  STRINGOUT1:VECT←VECT,ROS[N;]
[4]  N←N+1
[5]  →(N=11)/0
[6]  →STRINGOUT1
      ∇

```

Documentation

- * Input is a 10 by N matrix ROS
- * Output is a vector VECT consisting of the rows of ROS

```

      ∇ RND[ ] ∇
∇ R←N RND X
[1]  R←(10*-N)×[0.5+X×10*N]
      ∇

```

Documentation

- * Library defined rounding function
- * R is a number equal to X rounded to N places of decimal

A- 2 Main Scheduling Programs

A-2a. Main Routine

```

      ∇MAIN[ ]∇
    ∇ MAIN
[1]   MAT←(14 9)ρ0
[2]   'ENTER DURATIONS CAPACITIES AND NUMBER OF JOBS'
[3]   INPDUR←□
[4]   INPCAP←□
[5]   INPNRJOB←□
[6]   'SPECIFY SHOP'
[7]   SHOP←□
[8]   RANDOMS
[9]   MAT[1;]←SUM
[10]  RANDOMFITS
[11]  MAT[2;]←SUM
[12]  ( ' ' )
[13]  SSRANDOMFIT
[14]  MAT[3;]←SUM
[15]  LSRANDOMFIT
[16]  MAT[4;]←SUM
[17]  SPFITS
[18]  MAT[5;]←SUM
[19]  LPFITS
[20]  MAT[6;]←SUM
[21]  ( ' ' )
[22]  SPSCFITS
[23]  MAT[7;]←SUM
[24]  SPLCFITS
[25]  MAT[8;]←SUM
[26]  LPSCFITS
[27]  MAT[9;]←SUM
[28]  LPLCFITS
[29]  MAT[10;]←SUM
[30]  ( ' ' )
[31]  SSSPFIT
[32]  MAT[11;]←SUM
[33]  SSLPFIT
[34]  MAT[12;]←SUM
[35]  LSSPFIT
[36]  MAT[13;]←SUM
[37]  LSLPFIT
[38]  MAT[14;]←SUM
[39]  STATS
    ∇

```


Documentation (for MAIN on Page 6)

- * This is the main scheduling program
- * Accepts definition of the roster and shop from the keyboard and incorporates the scheduling routines. The following is automatic when the program MAIN is called.
- * Displays a ☐ and unlocks the keyboard four times. The following are defined by the operator in sequence:
 - INPDUR - roster durations vector
 - INPCAP - roster capacities vector
 - INPNRJOB - number of jobs in roster
 - SHOP - shop capacity vector
- * Executes each of the fourteen scheduling rules in turn with these global variables as input
- * Stores the summary characteristics calculated when each rule is implemented in the matrix MAT
- * Incorporates a program STATS which ranks the columns of MAT and prints the ranked summary matrix

A- 1.b Scheduling Programs

```

      ▽RANDOMMS[□]▽
    ▽ RANDOMS
[ 1]   P←INPDUR
[ 2]   X←INPCAP
[ 3]   NRJOB←INPNRJOB
[ 4]   C←SHOP
[ 5]   ASGNXT
[ 6]   SUMARY
    ▽

```

```

      ▽RANDOMFITS[□]▽
    ▽ RANDOMFITS
[ 1]   P←INPDUR
[ 2]   X←INPCAP
[ 3]   NRJOB←INPNRJOB
[ 4]   C←SHOP
[ 5]   ASGNAHD
[ 6]   SUMARY
    ▽

```

Documentation

- * Specifies the roster and shop as the global variables defined in MAIN
- * Incorporates the assignment routines ASGNXT for RANDOMS and ASGNAHD for RANDOMFITS
- * Incorporates the summary statistics generating routine SUMARY

```

      ▽SSSPFIT[□]▽
    ▽ SSSPFIT
[ 1]   CAP←INPCAP
[ 2]   DUR←INPDUR
[ 3]   NRJOB←INPNRJOB
[ 4]   ASSORT
[ 5]   DUR←AA
[ 6]   CAP←BB
[ 7]   ASSORT3
[ 8]   X←BB
[ 9]   P←AA
[10]   C←SHOP
[11]   ASGNAHD
[12]   SUMARY
    ▽

```



```

      ∇SSRANDOMFIT[□]∇
    ∇ SSRANDOMFIT
[ 1]  CAP←INPCAP
[ 2]  DUR←INPDUR
[ 3]  NRJOB←INPNRJOB
[ 4]  ASSORT3
[ 5]  X←BB
[ 6]  P←AA
[ 7]  C←SHOP
[ 8]  ASGNAHD
[ 9]  SUMARY
    ∇

```

```

      ∇SSLPFIT[□]∇
    ∇ SSLPFIT
[ 1]  CAP←INPCAP
[ 2]  DUR←INPDUR
[ 3]  NRJOB←INPNRJOB
[ 4]  DECSORT
[ 5]  DUR←AA
[ 6]  CAP←BB
[ 7]  ASSORT3
[ 8]  X←BB
[ 9]  P←AA
[10]  C←SHOP
[11]  ASGNAHD
[12]  SUMARY
    ∇

```

Documentation

- * Scheduling rules based on priority to smallest size job
- * Specifies the roster and shop as the global variables defined by MAIN
- * Performs the tie-breaking sort using ASSORT, nothing, and DECSORT to get smallest, random, or largest duration first
- * Uses the routine ASSORT 3 to arrange the roster in ascending size order
- * Specifies the input arguments for the scheduling routine ASGNAHD
- * Incorporates ASGNAHD and SUMARY to compute the summary statistics


```

      ∇ LSSPFIT[ ] ∇
    ∇ LSSPFIT
  [ 1]  CAP←INPCAP
  [ 2]  DUR←INPDUR
  [ 3]  NRJOB←INPNRJOB
  [ 4]  ASSORT
  [ 5]  DUR←AA
  [ 6]  CAP←BB
  [ 7]  DECSORT3
  [ 8]  X←BB
  [ 9]  P←AA
  [10]  C←SHOP
  [11]  ASGNAHD
  [12]  SUMARY
    ∇

```

```

      ∇ LSRANDOMFIT[ ] ∇
    ∇ LSRANDOMFIT
  [ 1]  CAP←INPCAP
  [ 2]  DUR←INPDUR
  [ 3]  NRJOB←INPNRJOB
  [ 4]  DECSORT3
  [ 5]  X←BB
  [ 6]  P←AA
  [ 7]  C←SHOP
  [ 8]  ASGNAHD
  [ 9]  SUMARY
    ∇

```

```

      ∇ LSLPFIT[ ] ∇
    ∇ LSLPFIT
  [ 1]  CAP←INPCAP
  [ 2]  DUR←INPDUR
  [ 3]  NRJOB←INPNRJOB
  [ 4]  DECSORT
  [ 5]  DUR←AA
  [ 6]  CAP←BB
  [ 7]  DECSORT3
  [ 8]  X←BB
  [ 9]  P←AA
  [10]  C←SHOP
  [11]  ASGNAHD
  [12]  SUMARY
    ∇

```


Documentation

- * Scheduling rules based on priority to largest size job
- * Specifies the roster and shop as the global variables defined by MAIN
- * Performs the tie-breaking sort using ASSORT, nothing, and DECSORT to get smallest, random, or largest duration first
- * Uses the routine DECSORT 3 to arrange the roster in descending size order
- * Specifies the input arguments for the scheduling routine ASGNAHD
- * Incorporates ASGNAHD and SUMARY to compute the summary statistics

```

      ∇ SPSCFITS[ ] ∇
∇ SPSCFITS
[1]  CAP←INPDUR
[2]  DUR←INPCAP
[3]  NRJOB←INPNRJOB
[4]  ASSORT
[5]  DUR←BB
[6]  CAP←AA
[7]  ASSORT
[8]  X←BB
[9]  P←AA
[10] C←SHOP
[11] ASGNAHD
[12] SUMARY
      ∇

```

```

      ∇ SPFITS[ ] ∇
∇ SPFITS
[1]  DUR←INPDUR
[2]  CAP←INPCAP
[3]  NRJOB←INPNRJOB
[4]  ASSORT
[5]  P←AA
[6]  X←BB
[7]  C←SHOP
[8]  ASGNAHD
[9]  SUMARY
      ∇

```



```

      V S P L C F I T S [ ] V
    V S P L C F I T S
[ 1]   C A P ← I N P D U R
[ 2]   D U R ← I N P C A P
[ 3]   N R J O B ← I N P N R J O B
[ 4]   D E C S O R T
[ 5]   D U R ← B B
[ 6]   C A P ← A A
[ 7]   A S S O R T
[ 8]   X ← B B
[ 9]   P ← A A
[10]   C ← S H O P
[11]   A S G N A H D
[12]   S U M A R Y
    V

```

Documentation

- * Scheduling rules based on priority to the smallest duration job
- * Specifies the roster and shop as the global variables defined by MAIN
- * Performs the tie-breaking sort using ASSORT, nothing and DECSORT to get the smallest, random or largest capacity first
- * Uses the routine ASSORT to arrange the roster in ascending duration order
- * Specifies the input arguments for the scheduling routine ASGNAHD
- * Incorporates ASGNAHD and SUMMARY to compute the summary statistics

```

      V L P S C F I T S [ ] V
    V L P S C F I T S
[ 1]   C A P ← I N P D U R
[ 2]   D U R ← I N P C A P
[ 3]   N R J O B ← I N P N R J O B
[ 4]   A S S O R T
[ 5]   D U R ← B B
[ 6]   C A P ← A A
[ 7]   D E C S O R T
[ 8]   X ← B B
[ 9]   P ← A A
[10]   C ← S H O P
[11]   A S G N A H D
[12]   S U M A R Y
    V

```



```

      ▽LPFITS[ ]▽
    ▽ LPFITS
[1]   DUR←INPDUR
[2]   CAP←INPCAP
[3]   NRJOB←INPNRJOB
[4]   DECSORT
[5]   P←AA
[6]   X←BB
[7]   C←SHOP
[8]   ASGNAHD
[9]   SUMARY
    ▽

```

```

      ▽LPLCFITS[ ]▽
    ▽ LPLCFITS
[1]   CAP←INPDUR
[2]   DUR←INPCAP
[3]   NRJOB←INPNRJOB
[4]   DECSORT
[5]   DUR←BB
[6]   CAP←AA
[7]   DECSORT
[8]   X←BB
[9]   P←AA
[10]  C←SHOP
[11]  ASGNAHD
[12]  SUMARY
    ▽

```

Documentation

- * Scheduling rules based on priority to the largest duration job
- * Specifies the roster and shop as the global variables defined by MAIN
- * Perform the tie-breaking sort using ASSORT, nothing and DECSORT to get the smallest, random or largest size first
- * Uses the routine DECSORT to arrange the roster in descending duration order
- * Specifies the input arguments for the scheduling routine ASGNAHD
- * Incorporates ASGNAHD and SUMARY to compute the summary statistics

A - 2c) Dependent Sorter, Scheduler and Summary Routines

A - 2c - I) Sorter Routines

```

      VDECSORT[ ]V
    V DECSORT
[1]  AA←BB←0p0
[2]  A←DUR
[3]  B←CAP
[4]  DECSORT1:C←I/A
[5]  →(C=0)/DECSORT2
[6]  D←A1C
[7]  AA←AA,C
[8]  BB←BB,B[D]
[9]  A[D]←0
[10] →DECSORT1
[11] DECSORT2:→0
    V

      VASSORT[ ]V
    V ASSORT
[1]  AA←BB←0p0
[2]  A←DUR
[3]  B←CAP
[4]  ASSORT1:C←I/A
[5]  →(C>10000)/ASSORT2
[6]  D←A1C
[7]  AA←AA,C
[8]  BB←BB,B[D]
[9]  A[D]←1000000000
[10] →ASSORT1
[11] ASSORT2:→0
    V

```

Documentation

- * Input is vectors DUR and CAP
- * Program specifies a vector AA corresponding to DUR and a vector BB corresponding to CAP
- * AA is the vector DUR rearranged in descending order by DECSORT and ascending order by ASSORT
- * If the element DUR [I] becomes AA [J] , then the element CAP [I] will become BB [J] for all elements


```

      ∇ DECSORT3[ ] ∇
∇ DECSORT3
[1]  AA←BB←S←0ρ0
[2]  A←DUR
[3]  B←CAP
[4]  SIZE←A×B
[5]  DECSORT32:X←⌈ /SIZE
[6]  →(X=0)/DECSORT31
[7]  IND←SIZE\X
[8]  S←S,SIZE[IND]
[9]  AA←AA,A[IND]
[10] BB←BB,B[IND]
[11] SIZE[IND]←0
[12] →DECSORT32
[13] DECSORT31:→0
∇

```

```

      ∇ ASSORT3[ ] ∇
∇ ASSORT3
[1]  AA←BB←S←0ρ0
[2]  A←DUR
[3]  B←CAP
[4]  SIZE←A×B
[5]  ASSORT32:X←⌊ /SIZE
[6]  →(X=10000)/ASSORT31
[7]  IND←SIZE\X
[8]  S←S,SIZE[IND]
[9]  AA←AA,A[IND]
[10] BB←BB,B[IND]
[11] SIZE[IND]←10000
[12] →ASSORT32
[13] ASSORT31:→0
∇

```

Documentation

- * Input arguments are vectors DUR and CAP
- * Generates a vector SIZE $[I] = \text{DUR } [I] \times \text{CAP } [I]$
DECSORT 3 sorts SIZE in decending order and ASSORT 3
sorts SIZE in ascending order
- * Output arguments are vectors S, AA, and BB corresponding to
SIZE, DUR and CAP respectively
- * If SIZE $[I]$ becomes S $[J]$, DUR $[I]$ and CAP $[I]$
become AA $[J]$ and BB $[J]$ respectively for all elements

A-2C-II Summary Routines

```

      ∇SUMMARY[ ]∇
∇ SUMMARY
[1]  T←+/(P×X)
[2]  U←CC×[ /F
[3]  EFF←T÷U
[4]  MS←(+/NUM)÷ρ NUM
[5]  SUM←9ρ 0
[6]  SUM[1]←MS
[7]  SUM[2]←[ /F
[8]  SUM[3]←(+/JOB)÷([ /F)
[9]  SUM[4]←EFF
[10] BEGIN←(( [ /F)-(4|[ /F))÷4
[11] SLACK←+/C[BEGIN+12×BEGIN]
[12] SUM[5]←1-SLACK÷CC×2×BEGIN
[13] SUM[6]←+/P×X×(NUM-1)
[14] SUM[7]←+/NUM
[15] VECTMS←(ρ NUM)ρ MS
[16] SUM[8]←((+/(NUM-VECTMS)*2)÷ρ NUM)*0.5
[17] SUM[9]←((+/NUM*2)÷ρ NUM)*0.5
[18] SUM
      ∇

```

Documentation

* Input arguments required are as follows:

NUM = vector of start times
 F = vector of finish times
 P = vector of job durations
 X = vector of job capacity requirements
 CC = capacity of a uniform shop at start
 JOB = number of jobs running each time period
 C = unused shop capacity each time period

* Computes a vector SUM consisting of the nine summary characteristics described in Section 4.2.2. CII .

* Displays the vector SUM

A - 2C. III Scheduling Routines

```

      VASGNXT[ ] V
V ASGNXT
[1]  HEADING
[2]  AVCAP←(+ / X) ÷ ρ X
[3]  JOB←500 ρ 0
[4]  N←J+1
[5]  CC←C[1]
[6]  NUM←F←NRJOB ρ 0
[7]  ASGNXT1: NN←N
[8]  ASGNXT2: →(C[N] < X[J]) / ASGNXT5
[9]  ASGNXT3: C[N]←C[N]-X[J]
[10] JOB[N]←JOB[N]+1
[11] N←N+1
[12] →(N < NN+P[J]) / ASGNXT3
[13] N←NN
[14] F[J]←N+P[J]-1
[15] NUM[J]←N
[16] DISPLAY
[17] J←J+1
[18] →(J > NRJOB) / ASGNXT4
[19] →ASGNXT2
[20] ASGNXT5: N←N+1
[21] →ASGNXT1
[22] ASGNXT4: ( ' ' )
[23] ASGNDETL
      V

```

Documentation

- * Input: X = vector of job capacities
 P = vector of job durations
 C = shop capacity in each time period
 NRJOB = number of jobs in the roster
- * Output: AVCAP= average job capacity
 NUM = vector of start times
 F = vector of finish times
 JOB = number of jobs running in each time period
 C = unused shop capacity in each time period
- * Jobs are assigned in order of occurrence in X and P as described in Section 4.2.2. CIII
- * A complete display of the schedule is made. To avoid this modify statements [1] and [16] to →2 and →17 respectively. The summary statistics only will then be printed


```

      ∇ ASGNAHD[ ] ∇
∇ ASGNAHD
[1]  HEADING
[2]  AVCAP ← ( + / X ) ÷ ρ X
[3]  CC ← C[1]
[4]  JOB ← 500 ρ 0
[5]  N ← J ← 1
[6]  NUM ← M ← F ← NRJOB ρ 0
[7]  ASGNAHD12 : NN ← N
[8]  → ( C[N] < X[J] ) / ASGNAHD14
[9]  ASGNAHD13 : C[N] ← C[N] - X[J]
[10] JOB[N] ← JOB[N] + 1
[11] N ← N + 1
[12] → ( N < NN + P[J] ) / ASGNAHD13
[13] N ← NN
[14] M[J] ← 1
[15] F[J] ← N + P[J] - 1
[16] NUM[J] ← N
[17] DISPLAY
[18] ASGNAHD14 : J ← J + 1
[19] → ( J > NRJOB ) / ASGNAHD15
[20] → ( M[J] = 1 ) / ASGNAHD14
[21] → ASGNAHD12
[22] ASGNAHD15 : → ( L / M = 1 ) / ASGNAHD16
[23] J ← 1
[24] N ← N + 1
[25] → ASGNAHD14
[26] ASGNAHD16 : ( ' ' )
[27] ASGNDETL
      ∇

```

Documentation

- * Input and output arguments have identical labels to those of ASGNXT and, therefore, this program is interchangeable with it in the scheduling routines
- * Assignment is attempted in the order the jobs appear in X and P, but a search ahead for a job which does not violate capacity restrictions is made as described in Section 4.2.2. CIII
- * A complete display of the schedule is made. To avoid this, modify statements [1] and [17] to →2 and →18 respectively.
- * The schedule shows job duration, capacity, start time, and finish time in that order. If the option is taken, only the summary characteristics of the schedule produced by SUMMARY are displayed


```

      VASGNDETL[ ]V
    V ASGNDETL
[1]   OCC←[ /F
[2]   ('JOBS IN PROCESS ' ;JOB[1OCC];)
[3]   ('UNUSED CAPACITY ' ;C[1OCC];)
    V

```

Documentation

- * Input vectors are: F - job finish times, JOB - number of jobs in process and C - unused capacity
- * Displays the number of jobs in process and unused capacity for the time the roster is in the shop

A-2a. Summary Ranking Routines

```

      VSTATS[ ]V
    V STATS
[1]   N←1
[2]   STATS2:MAX←[ /MAT[ ;N]
[3]   MIN←[ /MAT[ ;N]
[4]   DIFF←MAX-MIN
[5]   I←1
[6]   STATS1:MAT[I;N]←(MAT[I;N]-MIN)÷DIFF
[7]   I←I+1
[8]   →(I<15)/STATS1
[9]   N←N+1
[10]  →(N<10)/STATS2
[11]  MAT←10000×(4 RND MAT)
[12]  MAT
    V

```

Documentation

- * Input is matrix MAT of summary statistics as generated by MAIN
- * Performs the column by column ranking of MAT described in Section 4.2.2. e
- * Output is a matrix MAT consisting of the rankings in place of the summary statistics

A - 3 SUMMARY COMPUTATION PROGRAMS

A - 3a Summary Generators

```

      ∇ ASYN[ ] ∇
∇ ASYN
[1]  ATOTAL ← (14 60) ρ 0
[2]  MSDATOTAL ← (14 3) ρ 0
[3]  N ← 1
[4]  ASYN1: ATOTAL[N;] ← AF3[N;], AF4[N;], AF5[N;], AP3[N;],
      AP4[N;], AP5[N;]
[5]  MSDATOTAL[N;] ← MSD ATOTAL[N;]
[6]  ( ' ' )
[7]  ( ' ' )
[8]  ( ' ' )
[9]  ( ' ' )
[10] → (N=14) / 0
[11] N ← N+1
[12] → ASYN1
      ∇

```

```

      ∇ BSYN[ ] ∇
∇ BSYN
[1]  BTOTAL ← (14 60) ρ 0
[2]  MSDBTOTAL ← (14 3) ρ 0
[3]  N ← 1
[4]  BSYN1: BTOTAL[N;] ← BF3[N;], BF4[N;], BF5[N;], BP3[N;],
      BP4[N;], BP5[N;]
[5]  MSDBTOTAL[N;] ← MSD BTOTAL[N;]
[6]  ( ' ' )
[7]  ( ' ' )
[8]  ( ' ' )
[9]  ( ' ' )
[10] → (N=14) / 0
[11] N ← N+1
[12] → BSYN1
      ∇

```


▽CSYN[□]▽

```

▽ CSYN
[1] CTOTAL←(14 60)ρ0
[2] MSDCTOTAL←(14 3)ρ0
[3] N←1
[4] CSYN1:CTOTAL[N;]←CF3[N;],CF4[N;],CF5[N;],CP3[N;],
    CP4[N;],CP5[N;]
[5] MSDCTOTAL[N;]←MSD CTOTAL[N;]
[6] ( ' ' )
[7] ( ' ' )
[8] ( ' ' )
[9] ( ' ' )
[10] N
[11] →(N=14)/0
[12] N←N+1
[13] →CSYN1
▽

```

▽DSYN[□]▽

```

▽ DSYN
[1] DTOTAL←(14 60)ρ0
[2] MSDDTOTAL←(14 3)ρ0
[3] N←1
[4] DSYN1:DTOTAL[N;]←DF3[N;],DF4[N;],DF5[N;],DP3[N;],
    DP4[N;],DP5[N;]
[5] N
[6] MSDDTOTAL[N;]←MSD DTOTAL[N;]
[7] ( ' ' )
[8] ( ' ' )
[9] ( ' ' )
[10] ( ' ' )
[11] →(N=14)/0
[12] N←N+1
[13] →DSYN1
▽

```

<u>Program Name</u>	<u>Input</u>	<u>Output</u>
ASYN	AF3, AF4, AF5, AP3, AP4, AP5	MSDATOTAL
BSYN	BF3, BF4, BF5, BP3, BP4, BP5	MSDBTOTAL
CSYN	CF3, CF4, CF5, CP3, CP4, CP5	MSDCTOTAL
DSYN	DF3, DF4, DF5, DP3, DP4, DP5	MSDDTOTAL

* Input is rankings for each roster according to each measure of effectiveness

* Output is mean and standard deviation for each measure of effectiveness


```

      ∇MSD[ ]∇
∇ MSDA←MSD A;MEAN;STDEV;AA;I;S
[1]  MEAN←(+ / A) ÷ ρ A
[2]  STDEV←((+ / (A - (ρ A) ρ MEAN) * 2) ÷ ρ A) * 0.5
[3]  MSDA←(ρ A), MEAN, STDEV
[4]  □←MSDA
[5]  ( ' ' )
[6]  AA←(ρ A) ρ 0
[7]  I←1
[8]  MSD2:S←L / A
[9]  →(S=1000000) / MSD3
[10] AA[I]←S
[11] A[A;S]←1000000
[12] I←I+1
[13] →MSD2
[14] MSD3:□←AA
      ∇

```

Documentation

- * Required input is a vector A.
- * Computes the order, mean and standard deviation of A and places the result in a vector MSDA
- * Displays MSDA
- * Arranges the vector A in ascending order in AA
- * Displays AA
- * This program is incorporated in the programs (A, B, C, or D) SYN which specify the vector A as each row of the matrix formed by joining (A, B, C, or D) F3, F4, F5, P3, P4, P5. MSDA then replaces, column by column, the zeros in the zero matrix MSD (A, B, C, or D) TOTAL


```

      ∇COMPARE[ ]∇
∇ COMPARE
[1]  N←1
[2]  M←1
[3]  COMPARE2:('COMPARING ';N;' AND ';M;)
[4]  MSDATOTAL[N;]DIFFMEANS MSDATOTAL[M;]
[5]  (')
[6]  M←M+1
[7]  →(M=15)/COMPARE1
[8]  →COMPARE2
[9]  COMPARE1:N←N+1
[10] →(N=15)/0
[11] (')
[12] (')
[13] M←N
[14] →COMPARE2
∇

      ∇DIFFMEANS[ ]∇
∇ A DIFFMEANS B;N1;N2;S1;S2;X1;X2;SHAT;SD;T;Z
[1]  N1←A[1]
[2]  X1←A[2]
[3]  S1←A[3]
[4]  N2←B[1]
[5]  X2←B[2]
[6]  S2←B[3]
[7]  Z←N1+N2-2
[8]  SHAT←((N1×S1×S1)+(N2×S2×S2))÷Z)*0.5
[9]  SD←SHAT×(((N1+N2)÷(N1×N2))*0.5)
[10] T←(X1-X2)÷SD
[11] →(T≥0)/DIFFMEANS1
[12] T←-T
[13] DIFFMEANS1:('T IS ';T;' WITH ';Z;' DF')
∇

```

Documentation

- * DIFFMEANS performs the t test of Section 4.2.3 on input vectors A and B and displays the result
- * COMPARE specifies A and B for DIFFMEANS to be each possible pairs of rows in MSD (A, B, C or D) TOTAL
- * The output is a statement of the row numbers of MSD (A, B, C or D) TOTAL being compared and the corresponding t value


```

      VRANK[ ]VR
    VRANK
[1]  N←1
[2]  INDEX←0p0
[3]  RANK3:VECTOR←AF3[N;],AF4[N;],AF5[N;],AP3[N;],AP4[N;],
      AP5[N;]
[4]  INDEX←0p0
[5]  RANK1:A←[ /VECTOR
[6]  →(A=1000000)/RANK2
[7]  IND←9+(VECTOR,A)
[8]  INDEX←INDEX,((IND-(10|IND))÷10)
[9]  VECTOR[IND-9]←1000000
[10] →RANK1
[11] RANK2:□←INDEX
[12] ( ' ' )
[13] →(N=14)/0
[14] N←N+1
[15] →RANK3
      VRANK

```

Documentation

- * Input vector is row by row from matrices (A, B, C, or D) F3, F4, F5, P3, P4, P5
- * Each row is rearranged in ascending numerical order and the index each element had in the original vector is recorded. The indices are computed so a first number 1 to 6 inclusive represents an element from F3, F4, F5, P3, P4 or P5 respectively. The second number if printed represents the position of the element in the matrix
- * Output is a display of these indices in the order the elements appear in the rearranged vector

APPENDIX B
CHARACTERISTICS OF ROSTERS - GENERATED MEAN 3
Uniform Distributed

#	<u>Capacities</u>		<u>Duration</u>		Total Work	Matrix Location ROS F3
	Mean	Std. Dev.	Mean	Std. Dev.		
10	2.88	1.34	2.6	1.36	194	1;
11	3.24	1.56	2.88	1.39	216	2;
12	3	1.44	3.2	1.47	245	3;
13	2.6	1.5	3.2	1.39	205	4;
14	2.88	1.37	2.48	1.42	187	5;
15	3.08	1.47	2.88	1.37	238	6;
16	2.96	1.28	3.04	1.4	213	7;
17	3.04	1.28	2.44	1.27	192	8;
18	2.96	1.59	2.52	1.55	203	9;
19	3.04	1.37	2.88	1.48	220	10;

Poisson Distributed

#	<u>Poisson Distributed</u>		Total Work	Matrix Location ROS P3
	Mean	Std. Dev.		
40	3.2	1.5	238	1;
41	3.12	1.58	206	2;
42	2.92	1.26	188	3;
43	2.96	1.11	226	4;
44	2.64	1.38	222	5;
45	3.28	1.34	227	6;
46	2.52	1.27	218	7;
47	3.4	1.26	228	8;
48	3.32	1.32	231	9;
49	2.92	1.32	199	10;

APPENDIX B
CHARACTERISTICS OF ROSTERS - GENERATED MEAN 4

#	<u>Capacities</u>		<u>Uniform Distributed</u>		Total Work	Matrix Location
	<u>Mean</u>	<u>Std. Dev.</u>	<u>Duration</u>			
			<u>Mean</u>	<u>Std. Dev.</u>		
20	4.2	1.98	3.6	1.96	369	ROS F4 1;
21	3.4	2.04	3.76	2.23	331	2;
22	3.56	1.9	3.56	1.96	295	3;
23	3.88	2.18	4.28	1.54	410	4;
24	4.28	2.27	3.8	1.85	400	5;
25	4.28	1.84	4.2	1.81	455	6;
26	3.44	2.02	3.32	2.13	275	7;
27	4.76	1.92	3.64	1.74	448	8;
28	2.8	1.52	4.64	1.81	331	9;
29	3.92	2.17	3.72	1.89	365	10;

<u>Poisson Distributed</u>				
				ROS P4
50	3.72	1.51	3.92	1.98
51	3.88	1.66	3.72	1.4
52	3.84	1.38	3.68	1.46
53	3.84	1.91	3.84	1.51
54	4	1.77	4.6	1.72
55	4.16	1.83	4.52	1.79
56	3.96	1.61	3.92	1.79
57	4.36	1.89	4.12	1.8
58	3.88	1.7	3.8	1.5
59	3.64	1.35	4	1.79
				352
				375
				351
				348
				485
				486
				403
				471
				359
				352
				1;
				2;
				3;
				4;
				5;
				6;
				7;
				8;
				9;
				10;

APPENDIX B

CHARACTERISTICS OF ROSTERS - GENERATED MEAN 5

Uniform Distributed									
#	Capacities		Duration		Total Work	Matrix Location	ROS	F5	Matrix Location
	Mean	Std. Dev.	Mean	Std. Dev.					
30	4.68	2.85	6	2.1	688				1;
31	4.92	2.42	4.44	1.88	525				2;
32	5.44	2.14	5.28	2.6	750				3;
33	4.68	2.66	4.36	2.59	502				4;
34	5	2.12	4.68	2.62	570				5;
35	4.44	2.67	5.24	2.47	632				6;
36	5.04	2.82	5.08	2.73	589				7;
37	4.44	2.64	4.84	2.62	480				8;
38	5.04	2.32	5.4	2.47	690				9;
39	4.2	2.65	4.88	2.64	526				10;
Poisson Distributed									
60	4.8	2.04	5.92	1.76	710		ROS	P5	1;
61	4.92	1.81	5.16	2.09	642				2;
62	5.52	2.21	5.4	1.81	783				3;
63	4.96	1.48	4.92	1.92	596				4;
64	5.04	2.2	4.36	1.65	546				5;
65	4.08	1.29	5.2	1.79	523				6;
66	5.2	1.98	5.12	1.99	687				7;
67	4.96	1.97	4.24	1.61	533				8;
68	5.72	1.4	5.04	1.43	703				9;
69	4.76	2.05	4.8	1.7	554				10;

